ARCHITECTURAL TECHNIQUES FOR DISTURBANCE MITIGATION IN FUTURE MEMORY SYSTEMS

by

SeyedMohammad SeyedzadehDelcheh

B.S., Shiraz Univ. of Technology, 2007

M.S., Iran Univ. Science and Technology, 2011

Submitted to the Graduate Faculty of

the School of Computing and Information in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

August 2018

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

SeyedMohammad SeyedzadehDelcheh

It was defended on

August 31th 2018

and approved by

Rami Melhem, School of Computing and Information

Alex K. Jones, Swanson School of Engineering

Youtao Zhang, School of Computing and Information

Jun Yang, Swanson School of Engineering

Feng Xiong, Swanson School of Engineering

Dissertation Director: Rami Melhem, School of Computing and Information

ARCHITECTURAL TECHNIQUES FOR DISTURBANCE MITIGATION IN FUTURE MEMORY SYSTEMS

SeyedMohammad SeyedzadehDelcheh, PhD

University of Pittsburgh, August 2018

With the recent advancements of CMOS technology, scaling down the feature size has improved memory capacity, power, performance and cost. However, such dramatic progress in memory technology has increasingly made the precise control of the manufacturing process below 22nm more difficult. In spite of all these virtues, the technology scaling road map predicts significant process variation from cell-to-cell. It also predicts electromagnetic disturbances among memory cells that easily deviate their circuit characterizations from design goals and pose threats to the reliability, energy efficiency and security.

This dissertation proposes simple, energy-efficient and low-overhead techniques that combat the challenges resulting from technology scaling in future memory systems. Specifically, this dissertation investigates solutions *tuned to particular types of disturbance challenges*, such as inter-cell or intra-cell disturbance, that are energy efficient while guaranteeing memory reliability.

The contribution of this dissertation will be threefold. First, it uses a deterministic counter-based approach to target the root of inter-cell disturbances in Dynamic random-access memory (DRAM) and provide further benefits to overall energy consumption while deterministically mitigating inter-cell disturbances. Second, it uses Markov chains to reason about the reliability of Spin-Transfer Torque Magnetic Random-Access Memory (STT-RAM) that suffers from intra-cell disturbances and then investigates on-demand refresh policies to recover from the persistent effect of such disturbances. Third, It leverages an encoding technique integrated with a novel word level compression scheme to reduce the vulnerability

of cells to inter-cell write disturbances in Phase Change Memory (PCM). However, mitigating inter-cell write disturbances and also minimizing the write energy may increase the number of updated PCM cells and result in degraded endurance. Hence, It uses multi-objective optimization to balance the write energy and endurance in PCM cells while mitigating inter-cell disturbances.

The work in this dissertation provides important insights into how to tackle the critical reliability challenges that high-density memory systems confront in deep scaled technology nodes. It advocates for various memory technologies to guarantee reliability of future memory systems while incurring nominal costs in terms of energy, area and performance.

Keywords: Technology Scaling, Inter/Intra Cell Disturbance, STT-RAM, DRAM, PCM.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

Memory Technology has kept pace with Moores Law over the past few decades and has reduced the cost per bit of memory through increasing the memory cell density and capacity. The basic building block for main memory in modern systems is Dynamic Random Access Memory (DRAM) that is built from a two-dimensional array of cells. It encompasses memory cells at the intersections of bitline pairs and wordlines. Unfortunately, DRAM is becoming limited by power and scalability challenges, thus, endangering the evolution of the memory system [Aggarwal et al.; David et al.; Kim]. Accordingly, alternative memory technologies that can either replace or augment DRAM need to be considered to build a large and reliable memory system. Amongst several memory candidates, both Spin-Transfer Torque Random Access Memory (STT-RAM) and Phase Change Memory (PCM) are emerging as promising technologies due to their desirable characteristics in terms of low access latency, non-volatility and negligible stand-by power.

An STT-RAM cell structure is composed of a Magnetic Tunneling Junction (MTJ) connected in series with a transistor. This cell is connected between the bitline and the source line whereas the wordline is responsible for switching off the transistor. An MTJ device consists of a reference (fixed) layer and a free layer, which are separated by an oxide barrier layer [Hosomi et al.]. In contrast, a Single Level Cell (SLC) PCM is programed by switching the chalcogenide material between a high resistance amorphous state (RESET) and a low resistance crystalline state (SET) through the application of a programming current [Kim and Ahn; Zhang and Li]. The large resistance contrast between the SET and RESET states enables the exploitation of partially crystallized states to store more than one bit per cell resulting in Multi Level Cells (MLCs). In current MLC PCM, the resistance range between the RESET and the SET states is split into four regions that represent the logic values '00',

'01', '10', and '11'.

Decreasing the feature size of each memory cell further reinforces a common phenomena among future memory systems, referred to as *disturbance*, that will negatively impact the memory reliability, energy efficiency and performance [Cha, 2011; Lee et al.; Mandelman et al.; Naeimi et al., 2013]. Unfortunately, recent studies have shown that the read disturbance rate that originates from intra-cell thermal interferences in STT-RAM [Naeimi et al., 2013] is growing with aggressive scaling and is going to be a major reliability issue in future technology nodes. While inter-cell thermal interferences in PCM cells was first observed at 54nm regime [Lee et al.], research on cell structures scaled below 22nm technology node [Ahn et al.; Kim et al., 2011] shows that they can negatively impact the system performance and energy efficiency.

In DRAM, vulnerability to wordline electromagnetic fluctuations exists in recent sub 40nm commodity chips due to physical limitations of process technology. When the cumulative voltage interference to the DRAM wordline becomes strong enough, the state of nearby cells can change leading to memory errors. Research [Kim et al., 2015, 2014] showed that through frequently alternating the charge of specific memory locations, voltage fluctuations can be used, intentionally, to affect the charge of adjacent cells [Gruss et al., 2016]. However, in addition to intentional malicious attacks [Aweke et al., 2016; Ghasempour et al., 2015], the unbalanced nature of some applications access patterns induce voltage fluctuations.

The current memory systems incur high energy, performance and area overhead to obtain a bare-minimum satisfactory reliability. The goal of my research is to achieve strong memory reliability and high energy efficiency while taking advantage of continued scalability with minimal hardware overheads in future memory systems.

1.1 SCALABILITY CHALLENGES IN FUTURE MEMORY SYSTEMS

When DRAM cells are scaled at sub-22nm nodes, coupling capacitances between wordlines can cause data retention problems [Cha, 2011; Mandelman et al.; Redeker et al.]. Due to capacitive coupling between memory cells on adjacent wordlines, voltage levels used while

accessing data on one wordline can affect data quality on non-accessed neighboring wordlines. This is the root of the voltage fluctuations in DRAM and is sometimes referred to as "***Inter-cell read disturbance***, or "***Wordline crosstalk***." To this end, this dissertation explores a low-cost hardware solution that deterministically tackles scaling-related wordline crosstalk.

Although, the recent STT-RAM technology does not suffer from inter-cell disturbances, its reliability can be affected by "***intra-cell read disturbance***" which is mainly caused by the limited thermal stability and accumulated read current pulses [Naeimi et al., 2013]. When a large current during read is applied, the intra-cell read disturbance accidentally flips the value stored within the MTJ cell resulting in an error that persists in subsequent reads until the cell is rewritten [Sun et al.]. While the intra-cell read disturbance is nearly negligible ($\approx 10^{-9}$) at 65nm, it will exceed 10^{-5} per bit read at 22nm technology node and will continue to increase as the technology node descends [Sun et al.; Zhang et al., 2015]. The traditional approaches took the mitigation of write errors and false reads into account via deploying error correcting codes (ECC). This dissertation relies on the error detection capability of ECC to mitigate intra-cell disturbances in addition to alleviating false reads and write faults.

Finally, memory systems based on PCM technology can also suffer from "***inter-cell write disturbances***" [Ahn et al.; Jiang et al.; Kim et al., 2011] when the bitline and wordline distances in PCM contract. Specifically, the heat used for resetting cells in the active wordline can bleed to neighboring cells (victim cells) in the same wordline or neighboring wordlines intensifying inter-cell write disturbances. Specifically, the generated heat reduces the resistance of the victim cell and may unintentionally change its chosen state. The sneak heat slowly decays vertically along the bitline while it diminishes fast horizontally along the wordline. Thus, likelihood of incidence of write disturbance errors along the bitline is more than that along the wordline. To highly diminish the incidence of bitline disturbance errors in SLC PCM, only inter-cell spacing along the bitline is reduced sacrificing memory capacity. However MLC PCM delivers high memory capacity at the expense of higher write energy compared to SLC PCM. Note that it is impractical to precisely program cells through a single pulse in MLC PCM; therefore, industrial prototypes and academic research resort to an iterative program-and-verify (P&V) policy [Nirschl et al.; Pantazi et al., 2009].

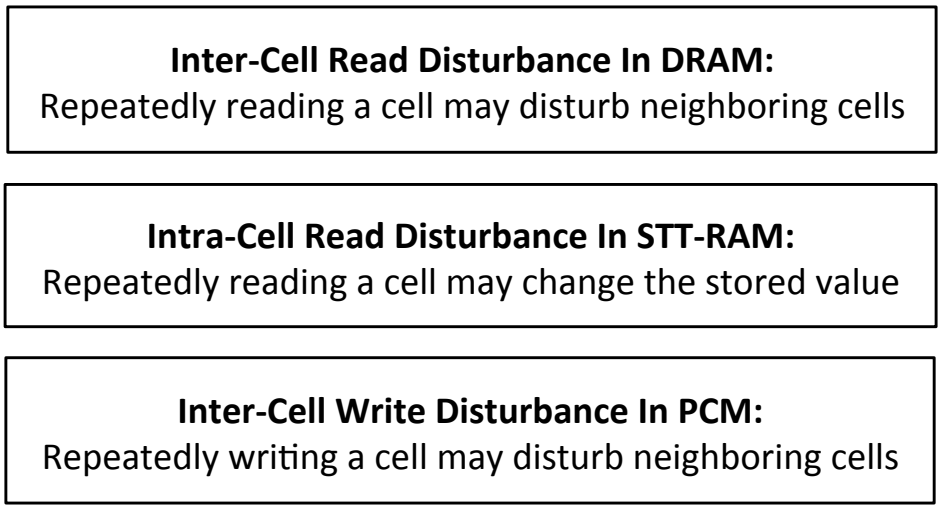


Figure 1: Scope of the dissertation.

Unfortunately, P&V programming increases the write energy by factors reaching 10× that of programming an SLC PCM [Wang et al.]. One of my objectives in this dissertation is to reduce write energy in MLC PCM by designing a simple and effective mechanism while still not adversely affecting susceptibility to inter-cell write disturbances.

1.2 RESEARCH OVERVIEW

The goal of my dissertation is to devise solutions that provide satisfactory energy efficiency in deep-scaled memory systems while achieving strong reliability via mitigating inter-cell/intra-cell disturbance errors.

The scope of the dissertation is depicted in Figure 1. Few effective solutions are available to support the poisonous challenges that these technologies face due to disturbances at deep scaling. In this realm, I am interested in finding effective solutions to the following research questions:

RQ1. How to design a low-overhead hardware structure that deterministically determines the most frequently access rows in DRAM and then mitigates the corresponding neighboring rows vulnerable to inter-cell read disturbances (wordline crosstalk)?

RQ2. How to tolerate both transient errors (false reads) and persistent errors (write faults and intra-cell read disturbance errors) in STT-RAM?

RQ3. How to manage the tradeoff among inter-cell write disturbances, energy and endurance in PCM while reducing the number of reset cells in order to minimize their impact on their neighboring cells?

In what follows, I elaborate on each of these research questions.

1.2.1 Mitigating Wordline disturbances in DRAM using Adaptive Trees of Counters

The conventional approach to mitigate wordline crosstalk in DRAM is to increase the refresh rate for all rows. Although, this approach is effective, it imposes an unnecessarily high power and performance overhead [Arjomand et al., 2016; Aweke et al., 2016; Kim and Papaefthymiou, 2003; Kotra et al.; Liu et al., 2012; Mukundan et al., 2013; Nair et al.; Ohsawa et al.; Rahmati et al.]. One hardware solution to mitigate wordline crosstalk in DRAM is to detect the most frequently accessed rows, or *aggressor rows*, and then refresh the rows that are adjacent to it, or *victim rows*. A simple method to recognize aggressor rows, called Static Counter Assignment (SCA), is to dedicate a counter per row to keep track of the number of row activations. However, having one counter per row induces a significant area and power overhead to the memory system. Due to row access locality in DRAM [Jeong et al.], many counters in SCA would be underutilized.

This dissertation proposes a Counter-based Adaptive Tree (CAT) approach that dynamically assigns counters to frequently accessed aggressor rows. Hence, with a small number of on-chip counters it is possible to deterministically refresh victim rows. When CAT results in a highly unbalanced tree, it provides a significant advantage in refresh energy over a block-based uniform counter distribution with a similar number of counters. In contrast, CAT converges to a balanced tree when accesses in memory are uniform. The key feature of CAT is that hot rows are instrumented using smaller groups, while rows with low access frequency are unlikely to induce crosstalk and are instrumented using larger groups.

1.2.2 Leveraging ECC to Mitigate Read Disturbances, in addition to False Reads and Write Faults in STT-RAM

The relatively unreliable reads of STT-RAM due to read disturbances degrades system reliability and precludes the integration of STT-RAM into the memory stack. The traditional ECC can detect latent read disturbances, as it does not differentiate between persistent errors (intra-cell read disturbances and write faults) and transient errors (false reads). Essentially, any error detected and corrected by ECC is treated as a potential read disturbance. Subsequently, either the data is written back or a second read is used to discover the nature of the error. Thus, a memory block is refreshed on demand upon error detection.

To this end, this dissertation first uses Markov modeling to build a strong understanding and characterization of how different types of errors and faults affect user operations. Then, based on this understanding, it studies low cost read disturbance policies that utilize the error detection capability of ECC to mitigate read disturbances. Finally, it takes advantage of the unique properties of the Markov chain process to estimate the reliability and overhead of the policies. Because of the cumulative effect of the read disturbance, even relatively low fault probabilities (raw bit error rate or RBER) can result in a relatively high probability of failure (unrecoverable bit error rate or UBER). Consequently, as Monte-Carlo simulation is only feasible for high RBER, it is inadequate for systems with persistent errors since it requires prohibitive simulation times to capture the effect of low RBER. This is the reason for using the proposed Markov Modeling in the evaluation.

1.2.3 Integrating Multi-Tiered Compression with Coset Coding for PCM to Mitigate Write Disturbances

This work integrates encoding techniques to compression techniques with the goal of using reclaimed bits for storing encoding meta data while salvaging memory capacity and making a trade-off among inter-cell write disturbances, write energy and endurance.

Typically, several adjacent data elements of a given size form a 64-byte cache line. For example, the cache line may encompass eight 64-bit double-precision floating-point values, sixteen 32-bit integers, or thirty-two 16-bit floating-point values. The significant similarities

in adjacent data elements stored in on-chip caches and off-chip memories have been observed in prior works [Kim et al., a; Yang et al., 2000; Zhang et al., 2000]. The main core of compression algorithms is based on exploring similarity among data elements. The similarity exploration can be conducted within data elements or across data elements. Unfortunately, the existing compressors change the bits in the data elements and do not allow the differential write to take advantage of *in-place similarity*[1] in PCM. The objective is to propose Multi-Tiered Compression (MTC) via exploring similarity within/across data elements to achieve the high percentages of compressed cache lines and reclaimed bits in PCM.

In contrast, the coset coding technique [Jacobvitz et al.; Seyedzadeh et al., 2016b] is an effective solution to minimize the cost function by expanding encoding space. It first maps each dataword to multiple coset candidates and then selects the coset candidate that minimizes the corresponding cost function. Finally, the selected coset candidate encodes the cacheline. To retrieve the original dataword in the decoder, the corresponding codeword is indexed by auxiliary bits that sacrifice the memory capacity. While encoding techniques work on the typical cacheline size to improve reliability, energy efficiency and endurance via sacrificing memory capacity, this dissertation revisits encoding techniques for PCM when lowering the encoding granularity below the typical cache line size.

1.3 THESIS CONTRIBUTION

This dissertation makes the following contributions:

1) For mitigating inter-cell read disturbances in DRAM:

- It demonstrates that, due to access locality in DRAM [Jeong et al.], instead of over-provisioning with one counter per row, a small number of counters can be implemented on-chip to refresh victim rows, while achieving low latency and low power consumption.
- It introduces a dynamically reconfigurable CAT scheme (DRCAT), that tracks and reacts to temporal changes in memory access patterns resulting from either appli-

[1] *In-place similarity* is the similarity of the old data to the corresponding new data in the memory line.

cation context switching or different phases of a particular application in order to more precisely identify actual victim rows and reduce DRAM refresh energy.

2) For mitigating intra-cell read disturbances in STT-RAM:

* It uses Markov chains to reason about the reliability of a system considering intra-cell read disturbances, write faults and false reads together and shows how an ECC can be used to cover the three different types of errors.
* It investigates three on-demand refresh policies to recover from the persistent effect of intra-cell read disturbances, false reads and write faults.

3) For mitigating inter-cell write disturbances in PCM:

* It characterizes realistic workloads and explores them for multi-tiered compression (MTC) that does not disturb in-place similarity to reduce write disturbance errors in deep scaled PCM while using very simple compression/decompression logic.
* It proposes a new and low overhead fine-grained restricted coset encoding that can be integrated with the proposed compression technique.
* It uses a multi-objective optimization technique to improve reliability, performance, write energy and endurance in deep scaled PCM.

1.4 ORGANIZATION

This dissertation is organized into six chapters. Chapter 2 reviews the related work. Chapter 3 implements adaptable trees of counters to alleviate wordline crosstalk in DRAM. Chapter 4 leverages ECC to mitigate read disturbance, false reads and write faults in STT-RAM. Chapter 5 investigates how to integrate coset coding with multi-tiered compression for PCM. Chapter 6 presents the summary and conclusion of the dissertation.

2.0 BACKGROUND AND RELATED WORK

This chapter first provides some necessary background on each memory technology and then briefly reviews related work.

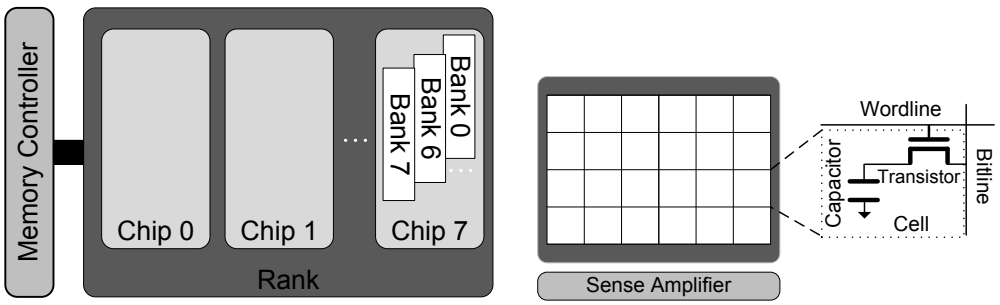
2.1 INTER-CELL READ DISTURBANCE (WORDLINE CROSSTALK)

2.1.1 DRAM Organization

DRAM-based main memory is a multi-level hierarchy of structures. At the highest level, each memory module is composed of a number of chips and is connected to the memory controller through a channel. Figure 2(a) shows 8 commodity DRAM chips that constitutes a typical rank. Internally, each chip consists of multiple banks and each bank is organized as rows of DRAM cells, as shown in Figure 2(b). Also, Figure 2(b) shows a single memory cell that is composed of a capacitor, in which the data is stored, and an access transistor. While accessing a row, all cells in the row are selected in parallel using a wordline. Due to capacitive coupling between cells on adjacent wordlines, if a wordline (aggressor row) is accessed frequently, voltage levels on neighboring wordlines (victim rows) can be affected leading to crosstalk. Mitigating wordline crosstalk is possible by refreshing the victim rows before the aggressor rows reach the refresh threshold.

2.1.2 Related Work

A hardware approach to alleviate wordline crosstalk is for each DRAM access to refresh the victim rows adjacent to the accessed row based on a probability function [Kim et al., 2015]. In this probabilistic approach, called PRA (Probabilistic Row Activation), the memory con-



(a) DRAM memory block.

(b) DRAM bank.

Figure 2: DRAM organization.

troller uses a Pseudo-Random Number Generator with a given probability (α) to determine when the memory controller should issue a refresh signal to refresh the two rows adjacent to the accessed row. When either the number of memory accesses or the probability α is high, this approach generates a significant number of refresh commands, thus exacerbating memory contention and increasing the energy cost [Aweke et al., 2016].

As a hardware alternative to the probabilistic approach, a deterministic approach can be used to prevent aggressor rows from being accessed more than the refresh threshold before refreshing the victim rows. Maintaining a counter for each memory row is a significant overhead [Bains and Halbert, 2016; Greenfield et al., 2015]. To address this problem, an approach was proposed that stores the counters in a reserved area of DRAM and a set-associative counter cache was established in the memory controller to improve accessibility to frequently used counters [Kim et al., 2015]. Note that the primary idea in [Kim et al., 2015] is similar to that used for Counter-based caches [Kharbutli and Solihin], where threshold-based counters detect expired lines for proactive eviction. While, using counters allows for accurate counts of row accesses, caching the counters introduces the complexity of maintaining a cache (e.g., tag matching, eviction policies) within the memory controller. Moreover, misses to the cache counter can be expensive.

Rewriting instructions, such as CLFLUSH [Seaborn and Dullien], have been proposed as software countermeasures against wordline crosstalk and are now deployed in Google Native Client. Similarly, access to the Linux pagemap interface is now prohibited from

userland [Shutemov, 2015]. These countermeasures have already been proven insufficient to mitigate malicious kernel attacks [Bosman et al.; Gruss et al., 2016]. In [Aweke et al., 2016], a generic software mechanism, ANVIL, is proposed to detect aggressor rows by monitoring the last-level cache (LLC) miss rate and row accesses with high temporal locality. A similar approach is proposed in [Herath and Fogh] to monitor the number of last level cache misses during a given refresh interval. Both approaches rely on software access to CPU performance counters.

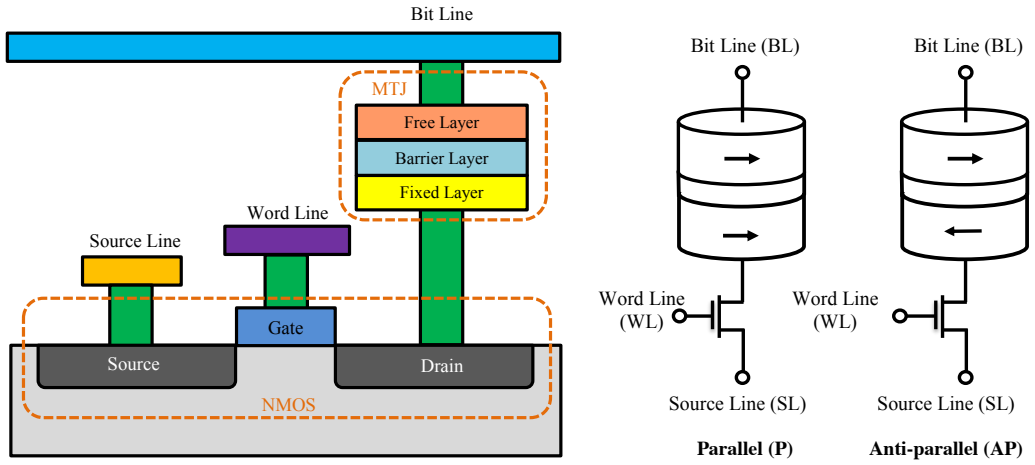
Our approach takes advantage of a small number of counters per bank, but better targets the aggressor rows to provide further benefits to overall energy consumption while deterministically mitigating crosstalk. Our novelty is the adaptive construction and dynamic reconfigurability of a "potentially unbalanced" tree of counters to match access patterns.

2.2 INTRA-CELL READ DISTURBANCE

2.2.1 STT-RAM

Figure 3(a) shows a cell structure of an STT-RAM composed of a Magnetic Tunneling Junction (MTJ) connected in series with a transistor. This cell is connected between the bit lines and the source lines whereas the word line is responsible for switching off the transistor. A MTJ device consists of a reference (fixed) layer and a free layer, which are separated by an oxide barrier layer [Hosomi et al.]. When the current flows from the free layer to the reference layer, the magnetization direction of the free layer flips to be *parallel* to that of the reference layer and the MTJ resistance becomes low representing a logical '0'; On the contrary, when the current is applied from the reference layer to the free layer, the magnetization direction of the free layer flips to be *anti-parallel* to that of the reference layer as shown in Figure 3(b). In this case, the MTJ resistance is high representing a logical '1'.

To write the data to the MTJ, a large current is injected so as to change the magnetic orientation of the free layer. The amount of the required current for writing into the MTJ is significantly larger than that for reading it. The read current can be injected into two



(a) STT-RAM cell.

(b) The equivalent circuit.

Figure 3: Spin-Transfer Torque Magnetic Random-Access Memory.

different directions, but the direction of writing '0' is always picked to increase reliability [Kawahara and et al]. The read current flowing in this direction potentially induces a unidirectional '1 $\rightarrow$ 0' flip.

2.2.2 Errors in read and write operations

In STT-RAM, write faults occur when the current is removed before the MTJ switching process completes [Wen et al.; Yang and et al, 2012]. False reads in STT-RAM are mainly caused by a decrease in the Tunnel Magneto-Resistance (TMR) ratio, and an increase in process variations in deep sub-micron technologies. If the current of parallel (anti-parallel) state in the MTJ crosses the threshold value of the anti-parallel (parallel) state, then the read returns a value different than the value stored within an MTJ resulting in a false read. Read disturbance is mainly caused by the limited thermal stability and accumulated read current pulses. When a large current during read is applied, the read disturbance accidentally flips the value stored within an MTJ cell resulting in an error that persists in subsequent reads until the cell is rewritten. The probability of a read disturbance (RBER) of an MTJ at a read current, I_r, is determined by the read current pulse width, τ, the thermal stability, Δ, and the critical switching current, I_0, as follows [Chen and et al, 2010; Koch and et al, 2004]:

$$p_d = 1 - exp\left(-\frac{\tau}{\tau_0} exp[-\Delta(1 - \frac{I_r}{I_0})]\right) \tag{2.1}$$

where, τ_0 denotes the thermally activated reversal time. Since the fabrication process determines the MTJ device parameters such as I_{c0} and Δ, they remain unchanged after a device is made. Therefore, the read disturbance probability is a function of $\frac{I_r}{I_0}$, under a given τ. Note that false reads are related MTJ thickness and TMR while disturbance and write errors derive from current densities [Yang and et al, 2012; Zhao and et al, 2012], so they are not tightly correlated.

In this work, we will denote the RBER due to false reads, read disturbances and write faults by p_f, p_d and p_w, respectively. Amongst false reads, read disturbances and write faults, only the effect of read disturbance is magnified with repeated read operations. Accordingly, the persistent nature of read disturbance requires special attention. Note that read disturbance errors in STT-RAM are local (intra-cell read disturbance), so they differ from crosstalk in flash memory and DRAM [Cai and et al; Cai et al.; Kim et al., 2014; Kultursay and et al]. For 180nm CMOS technology, the amplitude of the read current is much smaller than that of the write current and therefore read operations are reliable. Since CMOS technology continues to scale down, the amplitude of the read current and the write current are so close for 32nm technology node which dramatically increase the susceptibility of cells to read disturbance.

2.2.3 Related work

To tackle the read disturbance problem, several techniques have been proposed. A circuit based technique has been proposed for STT-RAM in which a pulsed read technique is used to read the content of the bit-cell [Raychowdhury]. In this technique, the word line switches to low and high states for a certain period of time to form a pulse that prevents the read current from flowing continuously through the bit-cell. Although this reduces the read disturbance rate, it increases read access time and the complexity of the sensing technique. At a device level, a disruptive reading and restoring scheme has been proposed in [Takemura and et al] to reduce the read disturbance rate by increasing the thermal stability factor. However, this

considerably increases the cycle time, the critical current and the write power.

A dual-mode architecture for fast-switching STT-RAM has been proposed in [Sun et al.] which can switch between two operation modes for either high data accuracy or low power consumption. In the high accuracy mode, the rewrite-after-read scheme is used to eliminate the data disturbances induced by the read current. To further reduce both dynamic and system energy consumption, a selective restore scheme has been proposed in [Rotenberg; Wang and et al]. This circuit-based scheme performs a double read operation with inverted read current to identify and restore all disturbed cells at the cost of a large read energy overhead.

SECDED (Single Error Correct, Double Error Detect) codes are used to recover from a single bit error in memory. If a data block already has an error, it is vulnerable to a second bit error that cannot be corrected. To prevent the occurrence of this second error, the memory is constantly examined in the background. When a single bit error is detected, it is corrected and the block is written back. This is referred to as Scrubbing [Awasthi and et al; Jacob and et al, 2010]. Memory systems with significantly high bit error rates require the development of scrubbing as an active defense against uncorrectable multi-failure errors. Moreover, scrubbing is predominantly valuable for systems where errors occur from external events to the storage cell (e.g., cross talk, SEU, etc.). In contrast, the errors we are modeling occur only from access to the modeled cell. Furthermore, with read disturbance, scrubbing should refresh/write the data even if no error is detected since reads from scrubbing may corrupt a cell, unnecessarily increasing write backs.

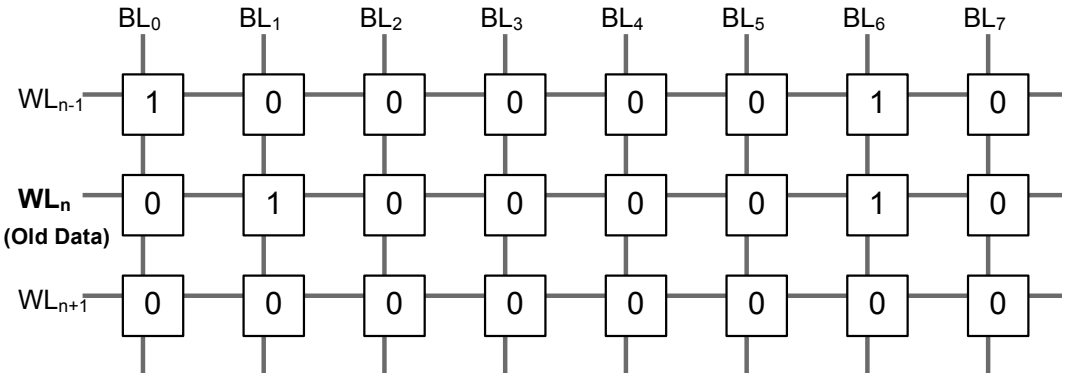
To mitigate various types of errors in STT-RAM, we investigate three policies which refresh the memory block on demand using a single ECC. According to the range of bit error rates, one of these three policies diminishes the destructive effects of disturbance errors. Also, we resort to Markov Chains to reason about the combined effect of persistent and transient errors on the UBER when on demand refresh is used to deal with read disturbance errors.

2.3 INTER-CELL WRITE DISTURBANCE

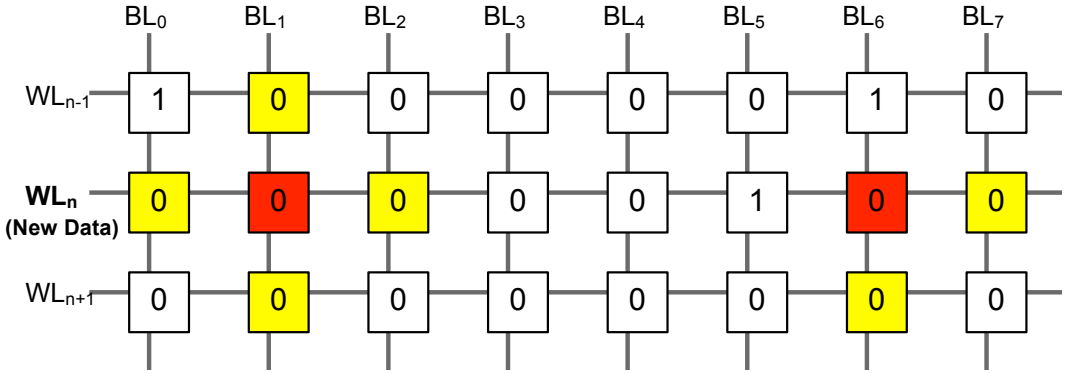
2.3.1 Single Level Cell PCM

A Single Level Cell (SLC) PCM is programed by switching the Chalcogenide material between a low resistance crystalline state (SET) and a high resistance amorphous state (RESET) through the application of a programming current. The cell in the SET state requires a high intense programming current to change its status to the RESET state. This current is considerably more than the current required for switching the cell from RESET to SET. Studies showed that the loss of cell endurance is directly correlated to the high programming current [Kim and Ahn; Zhang and Li].

Figure 4(a) shows the architecture of PCM cells, which are each deployed at the inter-



(a) The active wordline WL_n with old data values.



(b) The active wordline WL_n with new data values.

Figure 4: Write disturbance crosstalk in the deep scaled PCM cells. The red and yellow cells are aggressor and victim cells, respectively.

section of WordLines (WL) and BitLines (BL). Upon a memory write, some physical cells in the active wordline need to have their state changed (SET and RESET), while others remain untouched (*i.e.,* idle). Since the reset process produces significant heat, it can disturb cells that are idle. These idle cells that are inadvertently changed during the write process are *victim cells*. In this case, the resistance of a victim cell reduces and its logic value changes from '0 $\rightarrow$ 1'. The comparison of Figures 4(a) and 4(b) shows when a cell is reset '1 $\rightarrow$ 0', it can disturb cells within the active wordline WL_n and also in neighboring wordlines $\mathrm{WL}_{n\pm1}$. Potential victim cells vulnerable to crosstalk are represented in yellow color in Figure 4(b). For example, consider the cell, $c_{n,1}$, at the intersection of WL_n and BL_1. The value of $c_{n,1}$'s is updated from '1 $\rightarrow$ 0' during a new write. Since all the neighboring cells of $c_{n,1}$ are in the amorphous state, the heat resulting from the reset process may inadvertently update its neighboring cells with the probability 9.9% and 11% [Swami and Mohanram; Wang et al., 2015] in WL_n and $WL_{n\pm1}$, respectively. Note that the heat of aggressor cell does not have an effect on the cell that is in the SET state or is set during the write process. Thus, write disturbance crosstalk in PCM is asymmetric and unidirectional (*i.e.,* '0 $\rightarrow$ 1'). Note that if all cells are updated in a write operation, there are no idle cells and consequently no write disturbance. However, because the endurance of a PCM cell is determined by the number of writes to that cell, Differential Write [Zhou et al., 2009] is used and only cells which are actually changed are written.

2.3.2 Multi Level Cell PCM

To increase memory capacity in PCM, the large resistance contrast between the SET and RESET states enables to store more than one bit per cell (Multi-level cell). The typical MLC PCM [Nirschl et al.; Pantazi et al., 2009; Wang et al.] split the resistance range between the RESET and the SET states that represent the logic values '00', '01', '10', and '11'. Similar to a SLC PCM, the high heat resulting from resetting an MLC PCM, may disturb neighboring *idle* cells that are not being programmed. Specifically, the generated heat reduces the resistance of the idle cells and may unintentionally put them in the intermediate state or SET state with the probability that ranges from 12.3% to 27.6%. This reliability

bottleneck increases when memory cells are scaled below 22nm technology where cell-to-cell distance decreases considerably [Ahn et al.; Jiang et al.; Kim et al., 2011].

2.3.3 Related Work

Several techniques have been proposed to confront high write energy [Seyedzadeh et al., 2016a,b], endurance and write disturbance problems in SLC and MLC PCM. The key idea behind all of them is to reduce the number of state changes (write operations) that are costly in terms of energy, endurance and write disturbance. Data encoding is a common solution that effectively reduces the number of costly cell programming operations. For example, Flip-N-Write [Cho and Lee] was proposed for SLC PCM to reduce the number of written cells in the memory. To improve the lifetime of SLCs, FlipMin [Jacobvitz et al.] was proposed based on the concept of coset encoding [Forney]. The basic idea of FlipMin is to perform a one-to-one mapping from the data block to a coset of code word candidates. Then, the code word candidate that optimizes the lifetime is selected to be written in the memory. The initial coset candidates are built by the dual code of a (72,64) Hamming generator matrix. Since the initial coset candidates are essentially random binary vectors, FlipMin is most effective for workloads operating on random data [Seyedzadeh et al., 2016b].

To reduce write energy in MLC PCM and achieve low encoding overhead, an encoding that uses six coset candidates has been proposed in [Wang et al.] with the goal of mapping the two high energy states to the two low energy states. To mitigate word line write disturbance errors in wordlines, a Data encoding based INsulation technique (DIN), was proposed in [Jiang et al.] and was integrated with a 20-bit BCH code to correct any two write disturbance errors in a verification step. To make room for the extended code words, DIN uses memory line compression to compress a 512-bit line to 369-bits. However, because of the required large compression ratio, DIN is only able to compress and encode 30% of the memory lines.

To mitigate bitline write disturbance errors, SD-PCM uses idle error-correcting pointers (ECPs) [Wang et al., 2015] to temporarily recover from bitline disturbance errors in the adjacent memory rows. Using ECPs can postpone the extra write processes required for

neighboring rows if they are cached until the cacheline is evicted. SD-PCM leverages DIN to mitigate wordline disturbance errors. Because of increased cell activity of ECPs in SD-PCM, the availability of idle ECPs designed to handle hard errors like stuck-at faults limits the efficiency of SD-PCM. Furthermore, to avoid write disturbance errors in ECP cells (*i.e.,* to make the ECP bits safe from write disturbance), SD-PCM requires a lower-density ECP chip compared to memory chips for storing data.

Recently, ADAM [Swami and Mohanram] used Frequent Pattern based Compression (FPC) [Alameldeen and Wood, 2004; Pekhimenko et al., 2012] to reduce the number of cells written within a memory block. This naturally reduces the number of aggressor and victim cells in the active wordline. ADAM also reorganizes the compressed data such that consecutive memory rows store the compressed data in alternate alignments. *i.e.,* compressed data stored in even-numbered rows is right-aligned and compressed data in odd-numbered rows is left-aligned. Naturally, uncompressed data is stored as-is. Unfortunately, since FPC combined with base-delta immediate (BDI) compression [Pekhimenko et al., 2012] can only be applied in about 30% of cases [Seyedzadeh et al., 2018], ADAM is only effective for a small fraction of data blocks, limiting its overall effectiveness.

A low-area overhead technique is proposed in this dissertation that alleviates both bitline and wordline disturbance errors in SLC and MLC PCM. It opens room in the cacheline using a low-area overhead compression to store the auxiliary information of coset encoding. Due to not disturbing in-place similarity, the proposed technique reduces write energy, improves system performance while not sacrificing memory capacity.

3.0 MITIGATING WORDLINE CROSSTALK USING TREES OF COUNTERS

DRAM technology scaling has the undesirable side effect of degrading cell reliability. One such concern of deeply scaled DRAMs is the increased coupling between adjacent cells, commonly referred to as crosstalk. High access frequency of certain rows in DRAM may cause data loss in cells of physically adjacent rows due to crosstalk. The malicious exploit of this crosstalk by repeatedly accessing a row to induce this effect is known as *row hammering*. Additionally, inadvertent row hammering may also occur due to the natural weighted nature of applications' access patterns. This chapter analyzes the efficiency of existing approaches for mitigating wordline crosstalk and demonstrates that they have been conservatively designed. Given the unbalanced nature of DRAM accesses, a small group of dynamically allocated counters in banks can deterministically detect "aggressor" rows and mitigate crosstalk. Based on experimental findings, we propose a Counter-based Adaptive Tree (CAT) approach to mitigate wordline crosstalk using adaptive trees of counters to guide appropriate refreshing of vulnerable rows. The key idea is to tune the distribution of the counters to the rows in a bank based on the memory reference patterns. In contrast to deterministic solutions, CAT utilizes fewer counters, making it practically feasible to be implemented on-chip. Compared to existing probabilistic approaches, CAT more precisely refreshes rows vulnerable to crosstalk based on their access frequency.

3.1 MOTIVATION

This chapter analyses the previously proposed hardware approaches and makes key observations to motivate the dynamic counter assignment as a hardware solution that mitigates wordline crosstalk and combats row hammering.

3.1.1 Probabilistic Refresh Analysis

Using a probabilistic approach, such as PRA [Kim et al., 2015], to mitigate wordline crosstalk can protect against failure with a high probability, depending on the value of the refresh threshold, T, and the probability, α, of triggering a refresh. The probability of experiencing an error in Y years (defined as Y-years unsurvivability) for PRA is computed as:

$$unsurvivability = (1-p)^T \times Q_0 Q_1 \tag{3.1}$$

where $p = \alpha$ is the probability of refreshing TWO victim rows on an access, Q_0 is the number of refresh threshold windows during a refresh interval, and Q_1 is the number of 64ms periods during Y years. The parameter T depends on the technology node. Specifically, scaling down DRAM increases voltage fluctuations in cells because of the interaction between circuit components. Therefore, the refresh threshold is projected to decrease for future memory technology [Kim et al., 2015].

Figure 5 compares the 5-years unsurvivability for different refresh thresholds when p ranges from 0.001 to 0.006. Assuming mild row accesses during refresh intervals, we set Q_0 to 10, 15, 20, and 40. Figure 5 shows that for T=32K and $p > 0.001$, PRA's unsurvivability is lower than the Chipkill's unsurvivability of 1E-4. The key observation from this figure is that, for smaller values of T, larger values of p are needed to match the 5-years survivability of chipkill[1]. In fact, PRA's failure probability increases exponentially when the refresh threshold scales down, as is expected in future technology nodes. This means that larger values of p (more frequent random refreshes) are needed to guarantee acceptable survivability.

Note that the reliability reported in Figure 5 assumes the use of a true pseudo random number generator, PRNG, such as the one proposed in [Srinivasan et al.]. This is important since the computed reliability is contingent on the randomness of the numbers generated by PRNG. Specifically, the unsurvivability in Eq. 3.1 will not apply if a simpler (less costly in terms of area and power) PRNG is used since the randomness of the generated numbers will not be independent enough. To study the effect of the randomness of the generated numbers, we conducted a Monte-Carlo simulation to estimate the unsurvivability of PRA

[1]Similar analysis done in [Kim et al., 2015] shows that $PRA_{p=0.001}$ probability of failure is higher than 1E-4.

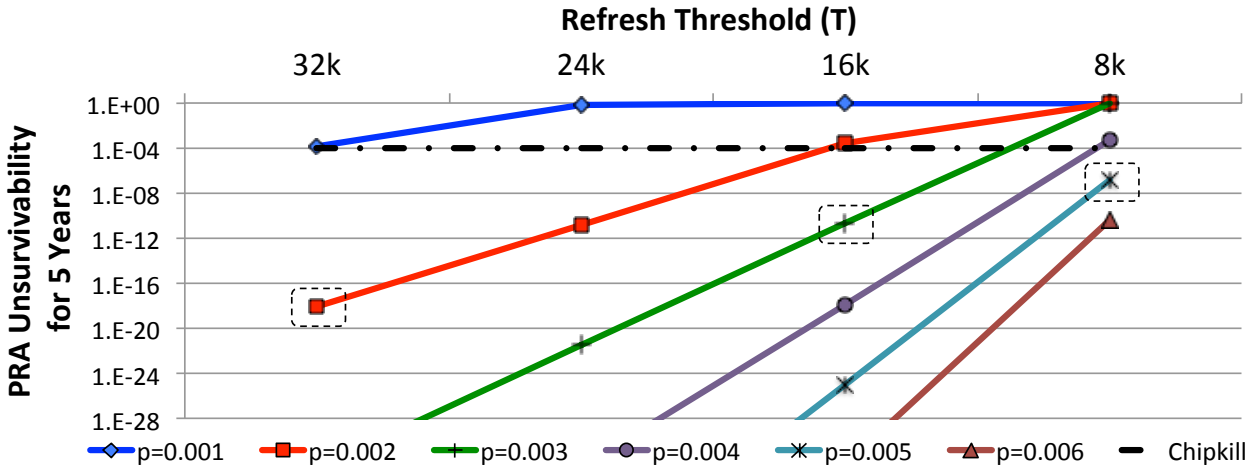


Figure 5: PRA unsurvivability for refresh thresholds 32k, 24k, 16k and 8k.

when a LFSR-based PRNG [lfs, https://users.ece.cmu.edu/ koopman/lfsr/] is used. The results show that, using an LFSR-based PRNG largely increases PRA's unsurvivability. For example, for T=16K and p=0.005, PRA's unsurvivability reaches 1E-4 after only 25 refresh intervals. To improve the reliability, a much larger value of p should be used with LFSR-based PRNGs which increases the refresh power and decreases the performance. A similar conclusion was reached in [Ghasempour et al., http://apt.cs.manchester.ac.uk/projects/ARMOR /RowHammer/ index.html]. Hence, PRA requires true random number generators, which are known to be complex and to consume relatively large power [Srinivasan et al.; Yang et al.], to achieve the probabilities shown in Figure 5.

3.1.2 Static Counter Assignment (SCA) Analysis

Using a deterministic approach for counting the number of accesses per row with on chip counters using SCA requires a large area and power overhead. One intuitive solution is to use fewer counters by partitioning the rows in each memory bank into fixed-size groups and assign one counter per group. To illustrate SCA, we assume that every bank in DRAM includes N rows and uses M counters. The refresh threshold, T, determines the size of every counter as $log_2 T$-bits. This approach, called SCA_M, divides the rows into M groups, each including $\frac{N}{M}$ rows. For every row activation, the row address maps to the appropriate

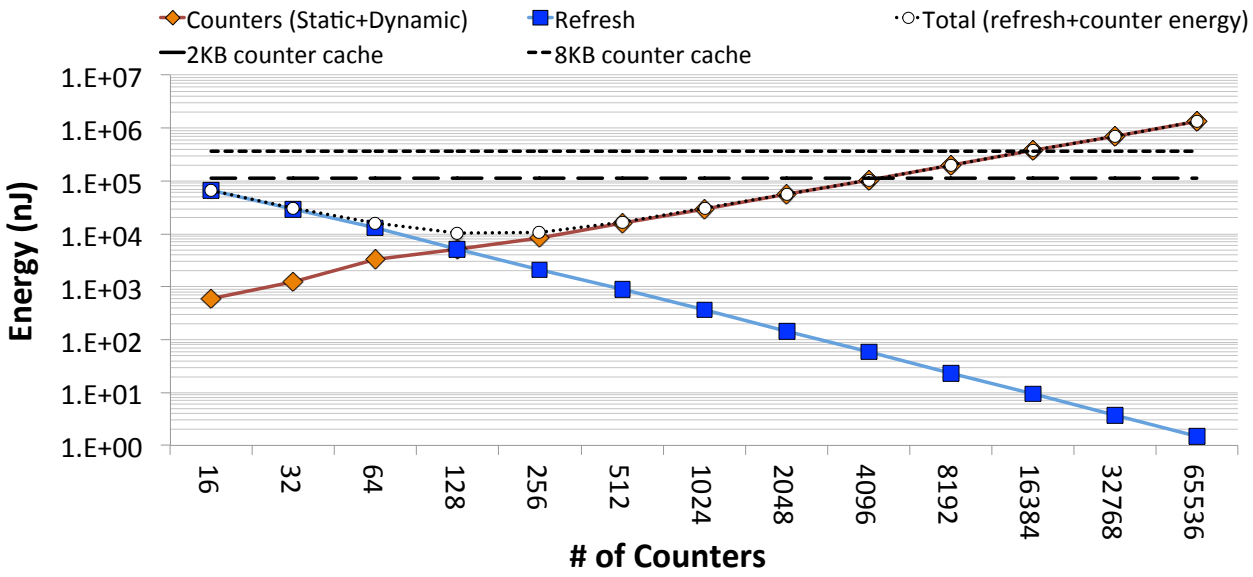


Figure 6: The energy overhead of SCA and counter caches [Kim et al., 2015] for different number of counters.

counter. Then, the corresponding counter counts the number of accesses. When the counter reaches the threshold, it is reset and a refresh signal is sent to the memory controller to refresh $\frac{N}{M}+2$ rows; the $\frac{N}{M}$ rows in the group plus the two rows adjacent to the group, which guarantees the refresh of any row in or adjacent to the group subjected to the crosstalk.

The energy overhead in SCA originates from activating the counters when memory is accessed and refreshing $\frac{N}{M}+2$ rows when a counter exceeds T. Figure 6 breaks down the energy overhead of SCA_M during a $64ms$ auto-refresh period when $N = 65536$ and the number of counters M ranges from 16 to 65536[2]. For a small number of counters, the energy resulting from refreshing victim rows (blue line) dominates the total energy of activating counters in SCA. In contrast, the total energy of activating counters in SCA is the dominant energy as the number of counters significantly increases (orange line).

Figure 6 shows that the total energy can be minimized at M=128. In this case, SCA_{128} not only reduces the energy overhead in comparison to SCA_{65536}, but also decreases the area overhead by two orders of magnitude (as will be explained in Section 3.5). In comparison,

[2]The refresh energy includes the average refresh energy of victim rows for 18 real workloads (Details in Section 3.4). We modified CACTI [Muralimanohar et al., 2009] to model the cache in the counter cache approach [Kim et al., 2015].

the counter cache approach [Kim et al., 2015], which stores counters in the reserved area of DRAM memory, reports data for a much larger counter storage cache, requiring capacity to store on the order of thousands of counters per bank. Ostensibly, this is to allow for enough flexibility to store the relevant counters to hot rows without a high miss rate due to capacity misses and/or thrashing. Thus, the energy overhead of counter storage cache will significantly exceed SCA_{128} due to the increased static power.

For example, Figure 6 shows the optimistic energy (assuming no misses requiring accesses to the DRAM) of 2K and 8K per bank counter caches as horizontal lines. These lines intersect the SCA_{4096}–SCA_{16384} points, respectively, as they have the same amount of total counter storage[3]. Thus, the total energy consumption of SCA with $M \leq 4K$ counters is lower than that of the counter caches with different sizes. In particular, *SCA_{128} can improve the total energy overhead and area overhead by 1.5 orders of magnitude in comparison to a 2K counter cache and nearly two orders of magnitude compared to an 8K counter cache [Kim et al., 2015].*

Thus, the key observation of these deterministic approaches is that allocating one counter to each row in a DRAM bank with a cache counter can be effective but are somewhat conservative and leave room for improvement. Specifically, the analysis of row access frequency of DRAM banks on real workloads reveals that the row access frequency during the refresh interval is not uniform and mostly a small group of rows are activated in DRAM banks. For example, Figure 7 depicts the row access frequency of a given bank for two typical real workloads, *blackscholes* and *facesim*, within a time period of one refresh interval ($64ms$). Figure 7 clearly shows that a small group of rows dominate overall accesses. *This motives us to propose a dynamic counter assignment for wordline crosstalk mitigation.*

[3]The counter caches have additional storage to store the tag array. However, this storage is typically less than the data array making it inconsequential on a log plot.

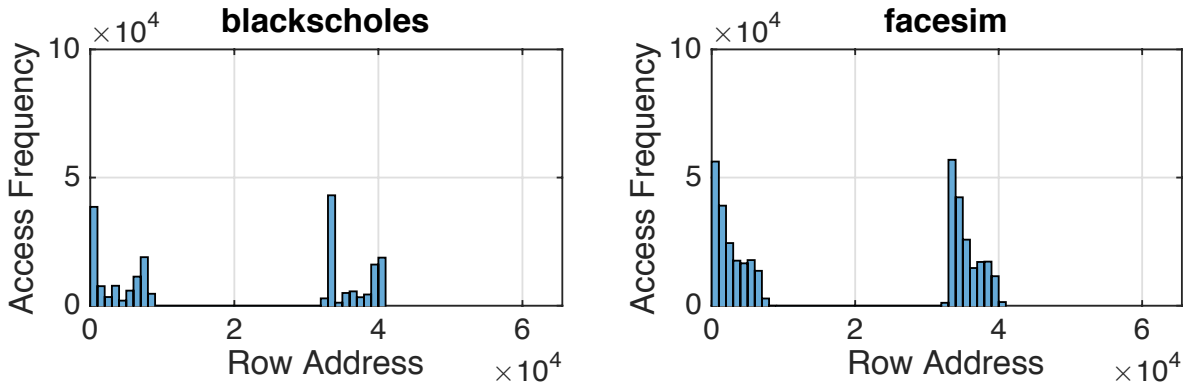


Figure 7: Row address frequency in a DRAM bank with 64K rows.

3.2 COUNTER-BASED ADAPTIVE TREE

In order to better assign row partitions to access counters, the Counter-Based Adaptive Tree (CAT) is a new and practical dynamic row partitioning technique that considers access frequency of rows to more carefully assign counters to appropriately sized groups of rows in order to improve energy and area efficiency. To divide an initial group of rows (e.g., a bank or some other uniform coarse partition) into groups of suitable sizes, CAT defines different split thresholds that identify access frequency stages prior to reaching the refresh threshold. These split thresholds are used to build a non-uniform binary tree structure that maps hot rows to smaller groups, while cold rows, *i.e.* rows with relatively low access frequencies, are mapped to larger groups. This aligns access counters to small groups of rows that contain an aggressor row to more precisely identify actual victim rows.

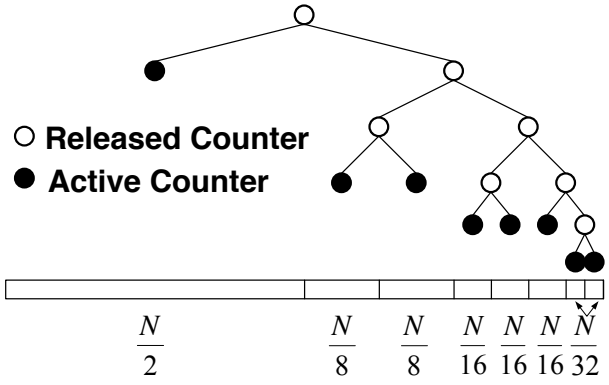
3.2.1 A simple CAT Example

Figure 8 depicts two trees built by CAT, where a terminal node represents an active counter and an intermediate node represents an expired counter, which had been split into two counters. The level of a node is defined as its distance from the root, with the root being at level zero. The levels of the CAT are associated with unique split thresholds. Hence,

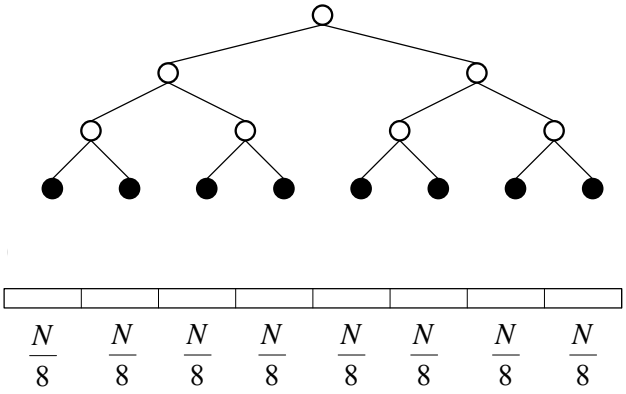
when a node reaches the next threshold, it further subdivides the group, or *splits* the node, generating two children counters initialized to the current count value. This is accomplished by activating a second counter as a clone of the existing counter. The binary tree of counters continues to grow until all available counters are activated or a maximum allowed level, a parameter of the CAT algorithm, is reached.

More precisely, assuming that we limit the number of levels in the tree to L, we define $L-1$ split thresholds T_0, ..., T_{L-2} where $T_0 \leq ... \leq T_{L-1}$ and $T_{L-1} = T$, recalling that T is the refresh threshold. Each of the M counters in a bank, C_0, ..., C_{M-1}, has $log_2 T$ bits and, initially, only C_0 is in active mode. When a counter at level l reaches the split threshold, T_l, it splits and two counters are activated at level $l+1$. This process continues until all the counters are activated or $l = L-1$. For example, Figure 8 shows two CATs for $L = 6$ and $M = 8$. The CAT in Figure 8(a) results from a non-uniform row access pattern, which causes more counters to be allocated to the hot row area (smaller blocks) and grows the tree through level 5. In contrast, when the row access frequency is uniform, counters are distributed uniformly throughout the bank addresses as shown in Figure 8(b). In this case, the CAT approach grows the tree only through level 3 and mimics SCA.

In CAT, N rows in one bank are initially treated as a group to which C_0 is allocated. As soon as C_0 reaches T_0, CAT splits C_0 into C_0 and C_1 with the same starting value of T_0. In this case, C_0 counts the number of accesses when the row address is between 0 to $\frac{N}{2}-1$ and C_1 counts the number of accesses when the row address ranges from $\frac{N}{2}$ to $N-1$. When C_1 reaches T_1, CAT splits C_1 into C_1 and C_2 with the new initial value T_1 where C_1 and C_2 track row addresses in the ranges from $\frac{N}{2}$ to $\frac{3N}{2}-1$ and from $\frac{3N}{2}$ to $(N-1)$, respectively. CAT continues this process until it activates all counters and no group can be split into smaller sub-groups. At this point, the split thresholds of counters are set to T. The minimum number of rows in a given group depends on the number of defined split thresholds. With $L-1$ split thresholds (a CAT with at most L levels), the minimum number of rows per group is $\frac{N}{2^{L-1}}$.



(a) Biased row address frequency.



(b) Uniform row address frequency.

Figure 8: The adaptive tress of counters for different workloads. The number of row addresses in the bank is N.

3.2.2 Constructing the CAT

Algorithm 1 shows the process for refreshing rows under the CAT structure per memory bank. It has two main modules: the Counter Module (CM) that records the number of row accesses and the Reconfiguration Counter Module (RCM) that activates and initializes counters. Assuming M counters in a given bank, CAT requires an array of M counter modules that are implemented on-chip, and one RCM that can be implemented either on-chip or in software. Each counter module CM_i maintains two registers, L_i and U_i to store the lower and upper row addresses assigned to this counter, and a register l_i to store the index of the split threshold used for that counter. The RCM maintains a *last_activated* counter register.

Initially, at the start of each refresh interval, CAT is reset such that only the first counter module, CM_0, is activated with L_0= 0, $U_0 = N-1$, l_0=0, and $last\_activated = 0$. Each time a row is accessed, its address is located in the range L_i - U_i of some active C_i, and this counter is incremented (lines 5-7). When C_i reaches T_{l_i}, $flag_i$ is raised (lines 8-10), which triggers RCM to activate a new counter as long as the number of active counters is less than M and the counter level $l_i < L-1$ (lines 15-16). When a new counter is activated, it is initialized by C_i (line 17) and the interval between L_i and U_i is split into two equal-size ranges where the lower bound of C_i remains unchanged and the upper bound of C_i is assigned to the upper bound of the new counter. Then, U_i shrinks to $U_i = \frac{U_i+L_i}{2}$ and the lower bound of the new counter is set to $U_i + 1$ (lines 18-20). The split thresholds of both counters are set to l_i+1

Algorithm 1: CAT structure per memory bank

```
1  Parameters: N : # rows per bank; M: # counters per bank; L: # thresholds; Input: row_address;
   Output: R_i: Refresh signal for refreshing all existing rows between L_i-1 and U_i+1.
2  begin
3      Counter Module CM_i /* i = 0, ..., M - 1 */
4      if L_i ≤ row_address ≤ U_i then
5          if C_i < T_{l_i} then
6              C_i ++;
7          else
8              if l_i < L - 1 then
9                  flag_i = 1; /* Signal to trigger RCM /*
10             else
11                 R_i=1; /* Signal to refresh corresponding rows*/
12                 C_i = 0;
13     Reconfiguration Counter Module (RCM) /* Activated when flag_i = 1*/
14     if flag_i == 1 for some i then
15         if last_activated < M - 1 && l_i < L - 1 then
16             last_activated++; /*Increase # of active counters*/
17             C_{last_activated} = C_i;
18             U_{last_activated}=U_i;
19             U_i = (U_i+L_i)/2;
20             L_{last_activated} = U_i + 1;
21             l_i ++;
22             l_{last_activated} = l_i;
23         if last_activated==M-1 then
24             for i=0:M-1 do
25                 l_i = L - 1;
```

(lines 21-22). For example, after initialization, when CM_0 reaches T_{l_0}, CM_1 is introduced by subdividing CM_0 in half, such that $C_1 = C_0$, $L_0 = 0$, $U_0 = \frac{N}{2} - 1$, $L_1 = \frac{N}{2}$, and $U_1 = N - 1$ with both CM's split thresholds being set to T_{l_1} and $last\_activated = 1$.

This process continues until some CM, CM_i, reaches the highest threshold $T_{l_i} = T$ (*i.e.*, if $l_i = L - 1$, lines 10-12). In this case, C_i is reset and the signal R_i is raised to cause the memory controller to refresh all existing rows in the address range of L_i-1 and U_i+1. When all counters are activated, CAT will set the index of all split thresholds to $l_i = L - 1$ which causes $T_{l_i} = T$ (line 25).

3.2.3 Efficient CAT Management Using SRAM

To directly implement Algorithm 1, maintaining the range boundaries of row blocks requires more storage than the actual counters, themselves. Given that SRAM uses less area and static power than registers [Jacob and et al, 2010], we are motivated to design and optimize the CAT for SRAM. In this case, instead of storing row range boundaries, we use pointers to *store the structure* of the CAT as shown in Figure 9. During each access, the tree structure is traversed sequentially by chasing the pointers to find the counter assigned to a specific row address.

The CAT, shown visually in Figure 9(a), is composed of two types of nodes: leaf nodes (shown in light blue) that represent active counters and intermediate nodes (shown in white) that determine the tree's structure. Rather than store all the nodes in our data structure, shown in Figure 9(b), we store only the intermediate nodes. Thus, we use an array, I, of size $M - 1$ (the maximum number of intermediate nodes in a tree with M leaves) to store information about intermediate nodes. Separately, we use another array C, of size M to store the counters, shown in Figure 9(c). For each intermediate node, two pointers, L_ptr and R_ptr, point to information about its two successors. If the successor is another intermediate node, L/R_ptr contains the entry for that intermediate node. If the successor is a leaf, L/R_ptr contains the entry for the counter corresponding to that leaf. Two flags, L_leaf and R_leaf, indicate if the corresponding successor is an intermediate or leaf node. The length of each counter is $log(T)$ bits and each pointer is $log(M)$ bits. The root of the tree, I_0, is deterministically stored in the first entry of the array I.

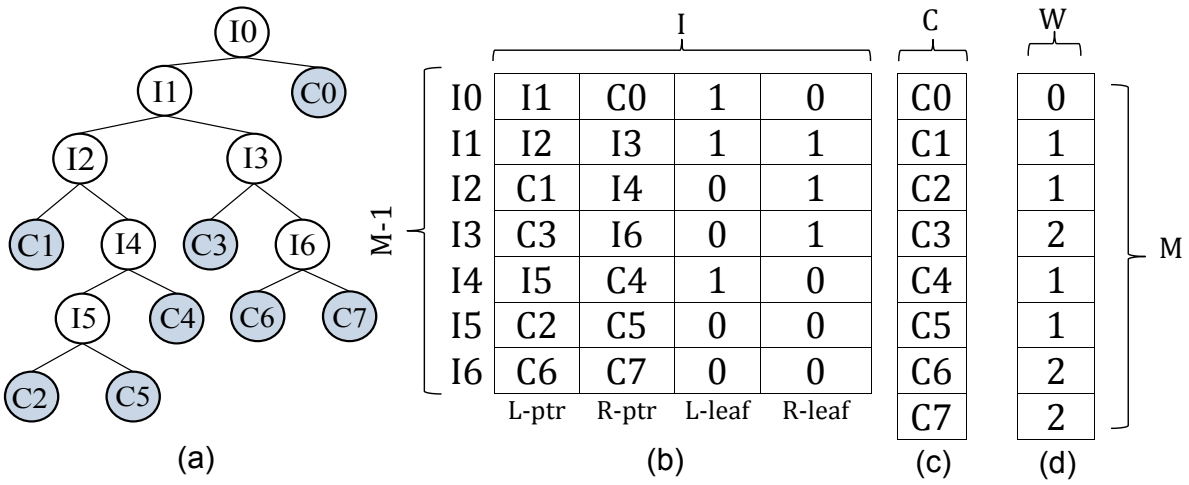


Figure 9: (a) A CAT using pointer chasing with M=8 counters and 6 levels. (b,c) The data structure used to represent the CAT. (d) An array of weight registers used for reconfiguring the CAT (see Section 3.3)

To determine whether to inspect the left or right entry, we examine the address A ($0 \leq A < N$). Starting at the root, the high order bit of A determines the successor that covers row A, thus accessing a leaf or an intermediate nodes as already described. More generally, when traversing an intermediate node at level l of the CAT, the l^{th} bit of A, counting from the most significant bit, is used to select the successor, which may be a leaf node or an intermediate node at level $l+1$. Before the CAT is completely built, it is guaranteed that fewer that M counters and $M-1$ intermediate nodes are utilized.

To illustrate the process of splitting a counter during the building of the CAT, we consider the example CAT articulated in Figure 9 by rolling back the last split operation. The last counter, C7 was deployed by splitting C6 into C6 and C7 and introducing I6. Let us assume that the current I6 still points to a leaf node C6. At this point of the CAT construction, only 7 counters were deployed. This incomplete tree is represented by the same array I of Figure 9 with the fourth entry of the array being [C3, **C6**, 0, **0**] (differences noted in bold) and the last entry being still undefined. To reach the state shown in Figure 9, C6 reached split threshold; I3's *R_ptr* was replaced by a pointer to the next available entry in I, I6; the last available counter, C7, is initialized to match C6; and I6 is set to [C6, C7, 0, 0] as is shown in the figure.

Given the above implementation, the maximum number of sequential SRAM accesses

for traversing the CAT is equal to the maximum depth of the tree, L. That number of accesses may be reduced if instead of starting to build the CAT tree from its root, we start from a pre-set complete binary tree with λ levels for some $\lambda \leq logM$. Consequently, to traverse the CAT, we can use the most significant λ bits of the row address, A, to directly access the appropriate intermediate node at level $\lambda - 1$, which reduces the maximum number of SRAM accesses to reach a leaf to $L - \lambda + 1$. For example, if we start from a uniform binary tree with $\lambda = logM$ levels, the initial CAT will be a complete tree containing $M/2$ counters and $M/2 - 1$ intermediate nodes. The other $M/2$ counters can then be used to grow the CAT non-uniformly beyond λ levels and up to a maximum of L levels. Moreover, by pre-splitting the counters uniformly up to level $\lambda - 1$ (that is starting from a balanced CAT with λ levels), we can reduce the size of the intermediate node array because we can avoid storing the intermediate nodes at levels smaller than λ.

3.2.4 Determining Split Threshold Values

The CAT adapts the distribution of the available counters to the rows in a bank depending on memory reference patterns. Specifically, the CAT is dynamically shaped to minimize the number of refreshed rows, and thus, the refresh power. Given a sequence of row references, the split thresholds determine the shape of the tree. In experimenting with the CAT technique, we found that its performance is sensitive to the values of the split thresholds. Given the combinatorial number of options for selecting the split thresholds, we present in this section a model to determine these thresholds in a way that minimizes the number of refreshed rows. We explain that model assuming that we start from a uniform CAT with $\lambda = m = logM$ levels, and determine the split thresholds $T_{m-1}, ..., T_{L-2}$ used to grow the tree non-uniformly to a maximum of L levels.

We consider first a simple example in which 4 counters are used for the N rows in a bank. Specifically, assume that after a number of references, the CAT is represented by the tree structure shown in Figure 10(a). Depending on the reference pattern and the values of the split thresholds T_1 and T_2, this structure can evolve to either the balanced tree structure of Figure 10(b) or the non-balanced tree structure of Figure 10(c). That is, whether counter

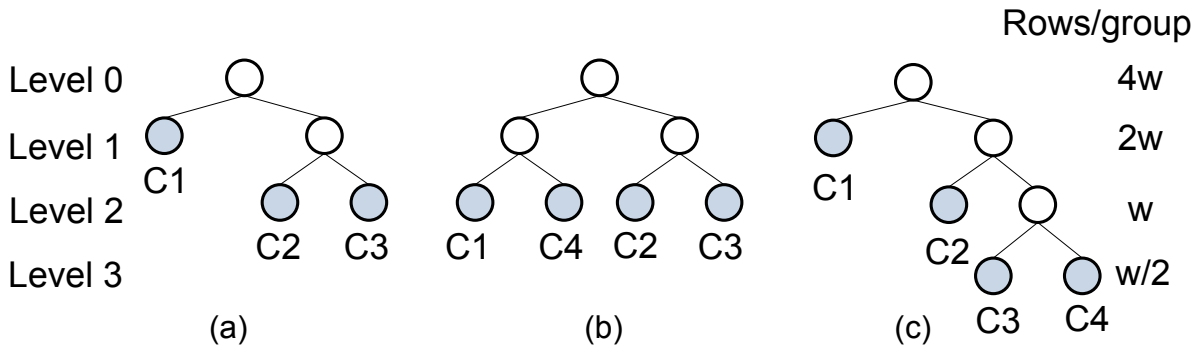


Figure 10: Two possible evolutions of the CAT of (a) to a balanced tree structure (b) or an unbalanced structure (c).

C1 splits first (Figure 10(b)) or counter C3 splits first (Figure 10(c)) depends on the relative value of T_1 and T_2. After one of the counters splits, the CAT reaches its final shape and the thresholds of all the counters are set to the row hammering threshold T. Hence, it is crucial to choose the split thresholds so that the CAT assumes the form of the tree that minimizes the number of refreshed rows.

Continuing with the 4-counter example, we note that a counter at level 0, 1, 2 and 3 will be assigned $4w$, $2w$, w and $w/2$ rows, respectively, where $w = N/4$. Now, assume that the bank receives R row references (accesses) during the regular refresh interval. If the references are uniformly distributed across rows, then each counter in Figure 10(b) will receive $R/4$ references, and if the row hammering threshold is T, then each counter will reach this threshold $R/4T$ times. Each time a counter reaches T, the w rows assigned to it are refreshed. Hence the total number of refreshes is

$$Cost_{SCA} = w \times R/T \tag{3.2}$$

Note that we should use $w+2$ instead of w in Eq. 3.2 to account for refreshing the two rows above and below each group of rows being refreshed. However, to simplify the discussion and formulas, we assume that $w >> 2$ and use w. Note also that even if the R references are not uniformly distributed among the rows, then $Cost_{SCA}$ is still expressed by Eq. 3.2 since, in this case, although the number of times each counter reaches T will be different for the four counters, the total number of times the four counters reach T will be R/T.

An unbalanced CAT is expected to reduce the number of refreshed rows if the R references are sufficiently biased towards a small group of rows. To determine the "amount" of bias that will favor the CAT of Figure 10(c) over the uniform tree of Figure 10(b), we define this bias using a variable x such that the group of rows assigned to counter C4 receives x more references than the other rows. That is the ratio of references caught by counters C1, C2, C3 and C4 is $2w : w : w/2 : x + w/2$. This means that each of the four counters will receive $r1 = 2w\alpha$, $r2 = w\alpha$, $r3 = 0.5w\alpha$ and $r4 = (x+0.5w)\alpha$ references, respectively, where $\alpha = R/(x + 4w)$. Consequently, the row hammering threshold T will be reached in counter C1 $r1/T$ times, and in each time the $2w$ rows assigned to it will be refreshed. Similarly, T in C2 will be reached $r2/T$ times, and in each time the w rows assigned to it will be refreshed. Finally, T in C3 and C4 will be reached $r3/T$ and $r4/T$ times, respectively, and in each time the corresponding $w/2$ rows will be refreshed. Hence, the total number of refreshed rows is

$$Cost_{CAT} = ((2w)^2 + w^2 + (\frac{w}{2})^2 + (x + \frac{w}{2})\frac{w}{2})\frac{\alpha}{T} \tag{3.3}$$

From Eq. 3.2 and Eq. 3.3, we conclude that $Cost_{CAT} < Cost_{SCA}$ when

$$x > 3w. \tag{3.4}$$

We call the value of $x = 3w$, the critical bias value. After determining the critical bias that causes the CAT to outperform the uniform tree, we proceed to find the thresholds T_1 and T_2 that will force, after a short sequence of accesses, say R_s, the tree of Figure 10(a) to evolve to the tree of Figure 10(c) if $x > 3w$ and to the uniform tree otherwise. For this, we note that if the reference bias is $x = 3w$, then after R_s references, the counters C1, C2 and C3 in Figure 10(a) will record $2w\beta$, $w\beta$ and $4w\beta$ accesses, respectively, where $\beta = R_s/7w$. Hence, if T_2 is set to be $2T_1$, then C3 will reach T_2 before C1 reaches T_1 when $x > 3w$, thus converging to the CAT of Figure 10(c). On the other hand, if $x < 3w$, then C1 will reach T_1 before C3 reaches T_2, thus leading to the uniform tree of Figure 10(b). To completely specify the split thresholds, we chose $T_2 = T/2$, which guarantees that the CAT converges before any counter reaches the threshold T. Consequently, $T_1 = T/4$.

Following the same reasoning as the 4-counter example, we consider the general case of $M = 2^m$ counters and the CAT shown in Figure 11 in which, due to reference bias, one

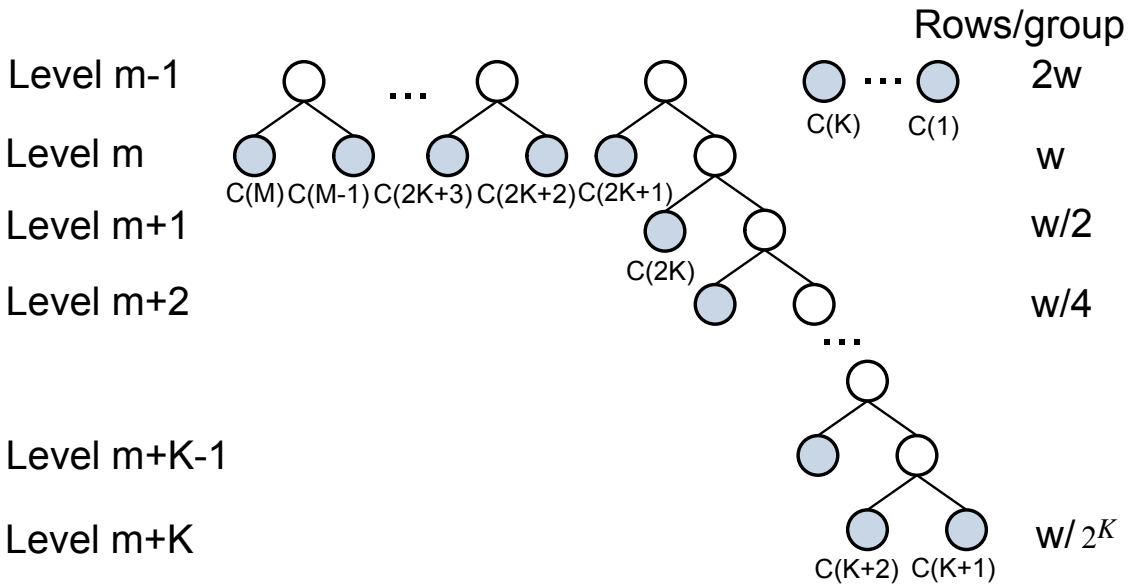


Figure 11: A CAT with one of the counters at level m spliting K times before K of the counters at level $m-1$ split.

counter, $C(2K)$, at level m has been repeatedly split $K = L-m-1$ additional times before any counter $C(1), ..., C(K)$, at level $m-1$, splits. If $w = N/M$ is the number of rows assigned to a counter at level m, then $w \times 2^{-k}$ is the number of rows assigned to each counter at any subsequent level, $m+k$. Note that K is bound by $log(w)$ because at $K = log(w)$, each group contains only $w = 1$ row.

We define a traffic bias of x_K to mean that one group of rows at level $m+K$, say the group corresponding to counter $C(K+1)$ in Figure 11, has $x_K + w \times 2^{-K}$ references rather than $w \times 2^{-K}$ references. To find the value of x_K that causes the CAT to have fewer refreshes than an m-level balanced tree, we note that for a total of R row references, each of the K counters at level $m-1$ receives $2w\alpha_K$ references, each of the $M-2K-1$ counters at level m receives $w\alpha_K$ references, counter $C(K+1)$ receives a biased traffic of $(x_K + w2^{-K})\alpha_K$ references and each of the other counters at level $m+j$, $j = 1, ..., K$ receives $2^{-j}\alpha_K$ references, where $\alpha_K = R/(x_K + wM)$. Dividing the value of each counter by T and multiplying by the size of the group assigned to this counter gives the number of refreshed rows as

$$Cost_{CAT} = (K(2w)^2 + (M-2K-1)w^2 + (\frac{x_K}{2^K} + \frac{w}{2^{2K}}) + \sum_{j=1}^{K}(\frac{w}{2^j})^2)\frac{\alpha_K}{T} \tag{3.5}$$

By comparing Eq. 3.5 with Eq. 3.2, which specifies the number of refreshes in the balanced tree (it is valid for any number of counters), we can calculate that the number of refreshes in CAT is fewer than in the balanced tree when

$$x_K > \left(\frac{6K + 2^{-2K+1} - 2}{3(1 - 2^{-K})}\right) w \tag{3.6}$$

Next, we consider a small number of references, $R_s < T$, and for a bias x_K, count the number of references, y_{m+K-1} seen by the counter at levels $m + K - 1$. Specifically,

$$y_{m+K-1} = (x_K + 2^{-K+1}w)\beta_K \tag{3.7}$$

where $\beta_K = R_s/(x_K + wM)$. Given that the number of references at any of the counters $C(1), ..., C(K)$ that did not split (at level $m - 1$) is $2w\beta_K$, then if we set the split threshold at level $m - 1$ to $T_{m-1} = 2w\beta_K$, then when the reference bias reaches the value specified by Eq. 3.6, we can force the CAT to grow from level $m+K-1$ to level $m+K$ before any of the counters at level $m - 1$ splits by setting $T_{m+K-1} = y_{m+K-1}$. Note that the split thresholds T_{m+K-1}, $K = 1, ..., L - m - 1$, are determined in terms of R_s, the number of references while the CAT is being formed. Specific values of the split thresholds can be set by choosing T_{L-2} to be a suitable fraction of the row hammering threshold, T. This determines R_s and consequently the other split thresholds. For example, using Eq. 3.7, the split thresholds of the tree with $M = 64$ counters and $L = 10$ levels can be computed to be $T_5 = 5155$, $T_6 = 10309$, $T_7 = 12886$, $T_8 = 16384$, and $T_9 = T = 32768$.

Finally, we note that the above model assumes that references are biased towards only one group of rows. A more complex model can be derived if references are biased towards more than one group of rows. However, our experimental study showed that with the split threshold determined by this simple model, the performance of CAT is much better than many of the other naive choices of thresholds, such as equally spacing them (different by a constant), setting them to values proportional to the tree levels or to values that double (or multiply by some fraction) at consecutive levels.

3.3 RECONFIGURING THE CAT TO TRACK CHANGES IN ACCESS PATTERNS

The CAT assigns the available counters to the rows of a bank according to the pattern of row accesses. However, the row access pattern changes with time, which necessitates a mechanism for the reconfiguration of the CAT to track these changes. In the next two sections, we propose two such mechanisms. The first, PRCAT, periodically reconstructs the CAT and the second, DRCAT, dynamically reconfigures the CAT by reassigning counters from cold to hotter regions of the bank.

3.3.1 Periodically Reset CAT (PRCAT)

In this scheme, the CAT tree is rebuilt at epochs equal to the auto-refresh interval (64ms for several DRAM generations [Chatterjee et al., 2012]). For LPDDRx devices that support burst refresh [Bhati et al., 2016], this simple scheme tracks the number of row accesses exactly. It can also be applied to modern DDRx devices that support distributed refresh at the expense of some inaccuracy in tracking the number of accesses. Specifically, because row refreshes are out of sync with the resetting of the CAT, recent information about row accesses are lost when the CAT is reset. Moreover, PRCAT resets the CAT periodically, even when the row access patterns do not change, potentially incurring the overhead of reconstructing the CAT unnecessarily. In the next section, we describe a CAT reconfiguration scheme which avoids these two shortcoming at a small cost for keeping additional information about the usage of the counters in a CAT.

3.3.2 Dynamically Reconfigured CAT (DRCAT)

The DRCAT allocates weights to counters to track the number of times each counter reaches the refresh threshold. After the CAT is completely built, the DRCAT identifies the counters allocated to regions that become cold and reallocate them to regions that become hot. A 2-bit weight register is used to record the weight of each counter. As described in the last section, when a counter reaches the refresh threshold, its corresponding rows are refreshed

and its value is reset to zero. However, to keep track of the hotness of row regions, the weight corresponding to that counter is incremented (with an upper bound of 3) and the weights corresponding to all other counters are decremented (with a lower bound of 0). If the weight of the incremented counter reaches its maximum limit, two counters having zero weights (cold regions) are merged and the released counter is used to split the hot counter.

To illustrate the scheme, consider the CAT example shown in Figure 9, where all counters have been activated and the weights of the counters are kept in the register W depicted in Figure 9(d). Assume that at a given time during operation, the values of the weight registers are [0,1,1,2,1,1,2,2] and counter C6 reaches its limit (we used 2-bit counters). After the rows corresponding to C6 are refreshed, the values of the weights are updated to [0,0,0,1,0,0,3,1] and the following steps are taken to reconfigure the CAT:

(1) From the table shown in Figure 9(b) an intermediate node in the CAT which has two counters as children (L_leaf = R_leaf = 0) with the weight of both being zero is selected. If such a node is found, the two counters are merged, one counter is freed and we go to step 2. In our example, C2 and C5 are leaves and both weight registers are zero. Hence, C5 is promoted to its parent node and the fifth row of the table is updated to I4=[**C5**,C4,**0**,0] as shown in Figure 12(b). Furthermore, C2 and the sixth row, I5, of the table are released.

(2) We split the region tracked by the hot counter using the counter freed in step 1. In our example, we show splitting C6 by replacing the L_ptr in its parent node (entry I6) by the index of the released row (I5) and set its corresponding flag to 1 to indicate that I5 will represent an intermediate node. Finally, we update I5=[**C6,C2,0,0**] to point to C6 and C2 and reset the corresponding flags.

(3) We update weight of the newly split counters to 1 to ensure they remain split for a reasonable period of time while preventing them from being quickly split in succession.

The DRCAT adds a negligible area overhead to the PRCAT design. For example, PRCAT uses 2 bytes per counter for T=16K and in this case, it occupies area overhead similar to DRCAT. The reason is that DRCAT uses the first 16 bits for the counter and the two last bits for the weight register. With respect to latency, the DRCAT traverses the tree to find

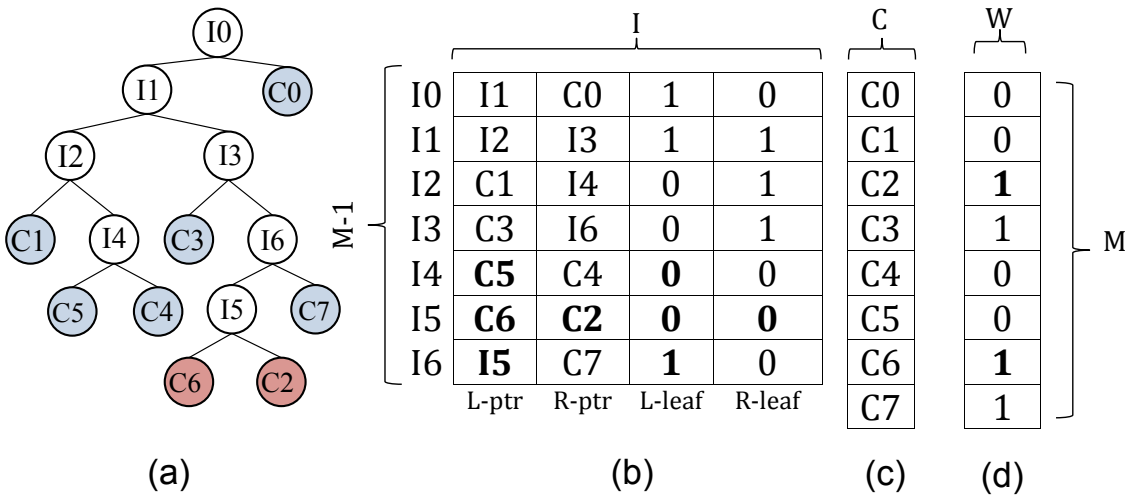


Figure 12: The CAT of Figure 9 after reconfiguration.

the cold counters and their parent intermediate node. Since the reconfiguration of the tree happens infrequently and traversing the tree is not on the critical path, system's performance is not affected by the reconfiguration.

Note that, in addition to tracking the change in the hot spot of memory accesses, the reconfiguration of the CAT according to the weights of the counters has the flexibility of adapting to multiple hot spots in the access patterns.

3.4 EXPERIMENTAL METHODOLOGY

To evaluate the proposed technique, we performed simulations using the memory system simulator USIMM modeling 55nm DRAM [Chatterjee et al., 2012]. Unless stated otherwise (in Section 3.6), the default simulation environment was set to model memory traffic from a dual core CPU. The total memory capacity is 16 GB with a total of 16 banks divided into two ranks, with 64K rows per bank. The last level cache size is 512KB per core in our simulation. Detailed simulation parameters for USIMM are listed in Table 1. The DRAM timing constraints follow a Micron DDR3 SDRAM data sheet [4Gb DDR3 SDRAM, 2011, 2011; Jacob and et al, 2010].

Table 1: System configuration

Processor	Two 3.2GHz cores, Memory bus speed: 800 MHz 128-entry ROB, Fetch width: 4, Retire width: 2 Pipeline depth: 10
Memory controller	Bus freq.: 800 MHz,Write queue capacity: 64 Address mapping: rw:rk:bk:ch:col:offset Management policy: closed-page with FRFCFS
DRAM	2 channels(each 8GB DIMM), 1 rank/channel 8 banks/rank, 64K rows/bank, 64B cache line

Verilog implementations of the control logic for the different wordline crosstalk mitigation schemes were created to provide an area and energy overhead comparison. These Verilog codes were synthesized using Synopsys Design Compiler and evaluated for power using Synopsys PrimeTime, targeting a 45nm FreePDK standard cell library [FreePDK45][4]. We have changed the number of counters per bank in the designs between 32 and 512 and, for CAT, allowed the trees to grow up to 14 levels to study the trade-off between performance, crosstalk mitigation refresh power, and hardware overhead. For the evaluation of PRA, we accounted for the energy to generate a random number every row access. To provide realistic workloads for evaluating the wordline crosstalk mitigation schemes, we used workloads from the Memory Scheduling Championship [Memory Scheduling Championship]. These workloads cover a variety of benchmarks including commercial applications and selected benchmarks from the PARSEC, SPEC, and Biobench suites. Furthermore, we use 12 kernel attacks to mimic malicious codes in Section 3.6.4.

One metric used to compare different crosstalk mitigation schemes is the crosstalk mitigation refresh power overhead (CMRPO). The CMRPO is the average power consumed for deciding which rows to be refreshed in order to mitigate crosstalk. It is computed relative to

[4]It is commonplace for DRAM to trail CMOS by a technology generation. Systems with 45nm CPUs were concurrent with 55nm DRAM.

the regular refresh power in the absence of any crosstalk mitigation (2.5mW to refresh 64K rows during a 64 ms refresh interval [4Gb DDR3 SDRAM, 2011, 2011; Rahmati et al.]).

While rows are refreshed in a bank to mitigate crosstalk, that bank cannot be accessed, possibly delaying subsequent memory requests to that bank. To estimate this delay, we define the execution time overhead (ETO) as the delay in execution time due to memory requests to banks being refreshed (to mitigate crosstalk) relative to the execution time when no provisions are made to mitigate crosstalk.

3.5 EVALUATION

We compare crosstalk mitigation schemes: PRA (refreshes two victim rows but not the aggressor row), DRCAT, PRCAT, SCA (implemented with SRAM). In this section, we conduct experiments on a dual-core system using refresh thresholds of T=32K and T=16K and a maximum of L=11 levels for DRCAT and PRCAT. In Section 3.6.1, we will study the effect of the maximum number of CAT levels and the value of the refresh thresholds on power and performance. Moreover, we will report results for quad-core systems. We assume that either the memory controller knows which rows are physically adjacent to each other [Van De Goor and Schanstra] or the DRAM chip is responsible for refreshing the row and its neighbors [Bains et al., 2016].

3.5.1 Hardware Overhead

Tables 2 and 3 show the hardware cost for managing and maintaining the counters for SCA, DRCAT and PRCAT with L=11 levels and $T = 32K$ as the number of counters per bank ranges from 32 to 512. We separately report the dominant sources of hardware energy overhead. These sources include: (1) the dynamic energy per access of the designed circuits plus the SRAM storage, and (2) the static energy during a 64 ms refresh interval of circuits plus the SRAM storage. The SRAM energy is extracted from CACTI [Muralimanohar et al., 2009] and the circuit energy (combinational and io-pad) is derived from Synopsys

Table 2: Hardware energy (per bank) of DRCAT, PRCAT and SCA for different number of counters, M.

M	Energy:dynamic (nJ per row access) and static (nJ per refresh interval)					
	DRCAT		PRCAT		SCA	
	dynamic	static	dynamic	static	dynamic	static
32	3.05E-04	5.77E+03	2.91E-04	5.55E+03	1.41E-04	3.16E+03
64	4.30E-04	1.39E+04	4.09E-04	1.32E+04	1.92E-04	8.81E+03
128	5.83E-04	2.77E+04	5.50E-04	2.63E+04	2.22E-04	1.44E+04
256	8.72E-04	5.44E+04	8.25E-04	5.13E+04	3.12E-04	2.39E+04
512	1.17E-03	1.06E+05	1.10E-03	1.02E+05	4.25E-04	4.52E+04

PrimeTime. Note that for DRCAT and PRCAT, the dynamic energy per memory access accounts for multiple accesses to SRAM (from 2 to $L - log(M/4)$) while for SCA, SRAM is accessed only twice to read and write the counters. A modified version of Tables 2 and 3 is used for DRCAT and PRCAT when the maximum tree depth changes in the experimental tests.

The results show that the dynamic energy per access of PRCAT is roughly twice that of SCA for the same number of counters. With respect to area overhead and static energy, Table 3 clearly shows that PRCAT and SCA occupy equal area and consume similar static power when the number of counters of SCA is twice that of PRCAT. For example, PRCAT_{64} and SCA_{128} occupy iso-area. Moreover, this area is one order of magnitude smaller than the area needed by the leading counter-based approach that stores in memory one counter per row and uses a 32KB on-chip counter cache [Kim et al., 2015] (equivalent storage to 2,048 counters per bank). Thus, implementing 64 or even 256 counters per bank is feasible. Our implementation shows that the average latency for PRCAT is $3.6ns$ (circuit latency plus repeated access to SRAM) which is much lower than the row activation latency in the DRAM memory [Shin et al., 2016].

In comparison to PRCAT for T=32K, DRCAT uses a 2-bit weight register per counter

Table 3: Area (per bank) of DRCAT, PRCAT and SCA for different number of counters, M and the specification of the PRNG used for PRA [Srinivasan et al.]. The reported energy for PRNG (eng_PRNG) is for generating 9-bits per row access.

M	Area(mm^2)			PRNG	
	DRCAT	PRCAT	SCA		
32	3.16E-02	3.04E-02	1.86E-02	Area(mm^2)	4.004E-3
64	6.12E-02	5.86E-02	4.04E-02	Throughput(Gbps)	2.4
128	1.16E-01	1.11E-01	6.04E-02	Power(mW)	7
256	2.23E-01	2.11E-01	1.00E-01	Eff.(nj/b)	2.90E-3
512	3.93E-01	3.75E-01	1.72E-01	eng_PRNG(nj)	2.625E-2

to reconfigure the structure of CAT. The results in Table 3 show that the circuit design and SRAM storage of DRCAT, on average, augments 4.2% area overhead to the system compared to PRCAT. Also, DRCAT increases the dynamic energy per row access by 5% over PRCAT. Furthermore, it incurs 4ns latency. When DRCAT reconfigures counters, its latency is about 7.5ns. The main reason for the extra latency is the traversal of the tree as explained in Section 3.3.2. However, updating the DRCAT and accessing the memory can be done in parallel.

Table 3 also shows the specification of a PRNG [Srinivasan et al.] for PRA in 45nm technology [5]. We select one PRNG for PRA that is applied for all banks during row accesses. The energy per bit (the efficiency) for PRNG is computed as $Power/Throughput$. For $p = 0.002$ and $p = 0.003$, PRNG generates 9 bits ($\sim log(1/0.003$ or $log(1/0.002)$) so that PRA can decide if victim rows should be refreshed when a row is accessed. The energy for generating 9 random bits is denoted by eng_PRNG. A similar conclusion was reached in [Ghasempour et al., http://apt.cs.manchester.ac.uk/projects/ARMOR /RowHammer/ index.html].

[5] An PRNG design with low static power is reported in [Yang et al.]. However, this design is much slower than the design in [Srinivasan et al.] which leads to a larger Energy/bit consumption.

3.5.2 CMPRO

We use the results shown in Tables 2 and 3 to compute CMRPO for a benchmark during its execution by adding the following components needed to mitigate crosstalk: (1) The dynamic power (product of dynamic energy per memory access and the total number of memory accesses during execution divided by the execution time), (2) the static power (static energy during a refresh interval divided by the refresh interval), and (3) the refresh power (product of the average number of rows refreshed to prevent crosstalk with the energy to refresh one row (1nJ per row [Ghosh and Lee]) divided by the execution time).

Figure 13 shows the CMRPO for different approaches when $T = 32K$. It reveals that both DRCAT_{64} and PRCAT_{64} with L=11 achieve a CMRPO of 4%, which is an improvement over the 11% in the cases of PRA and SCA. Note that the CMRPO for PRA includes refreshing an average of two victim rows every 500 accesses and generating 9 PRNG bits every access, with the PRNG generation being dominant. According to Table 3, on average, for every 50 row accesses, PRA consumes energy equal to that of refreshing one row.

For T=16K, we use $\text{PRA}_{0.003}$, rather than $\text{PRA}_{0.002}$ since the probability of failure for $\text{PRA}_{0.002}$ is greater than 1E-4 (Chipkill reliability) according to Figure 5. Figure 13 shows that CMRPO for DRCAT_{64} in dual-core systems is 4.5%, which is an improvement over the 12% and 22% incurred in $\text{PRA}_{0.003}$ and SCA_{64}, respectively. Also, considering iso-area, DRCAT_{64} achieves a CMRPO improvement over the 13% incurred in SCA_{128}. Figure 13 indicates that reducing T from 32K to 16K will increase considerably CMRPO for SCA while slightly increasing CMRPO for PRCAT and DRCAT.

3.5.3 Execution Time Overhead

To evaluate performance, we report the execution time overhead (ETO) resulting from refreshing victim rows. When rows vulnerable to crosstalk are refreshed, any read or write request to the bank containing the refreshed rows is stalled, which leads to the execution time overhead.

Figure 14 shows the ETO for different workloads. For $T = 32K$, $\text{PRA}_{0.002}$, SCA_{64}, SCA_{128}, PRCAT_{64} and DRCAT_{64} incur low ETO of 0.26%, 1.32%, 0.43%, 0.23%, and 0.16%

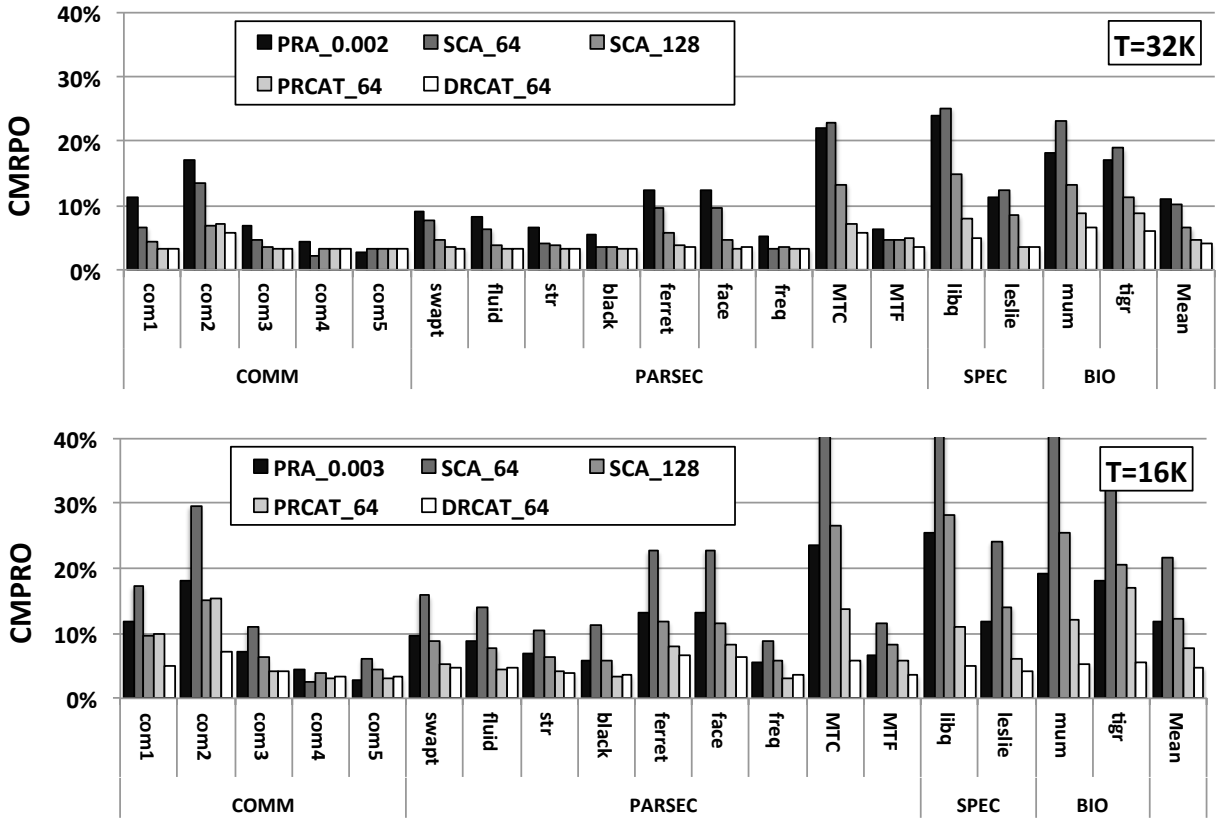


Figure 13: The CMRPO (as a percent of the regular refresh power). DRCAT and PRCAT use 64 counters and up to 11 levels.

respectively. For $T = 16K$, the ETOs of $\text{PRA}_{0.003}$, SCA_{64}, SCA_{128}, PRCAT_{64} and DRCAT_{64} are 0.39%, 3.42%, 1.38%, 0.49% and 0.35% respectively. Note that ETO for $\text{PRA}_{0.003}$ when $T = 16K$ is roughly 1.5 times larger than ETO for $\text{PRA}_{0.002}$ when $T = 32K$ because it probabilistically refreshes 50% more rows. On the other hand, ETO for SCA_{128} when $T = 16K$ is higher than ETO for SCA_{64} when $T = 32K$. This shows that when the refresh threshold is reduced, doubling the number of counters statically does not reduce the number of refreshed rows, which results in less accurate row tracking and thus larger refresh energy.

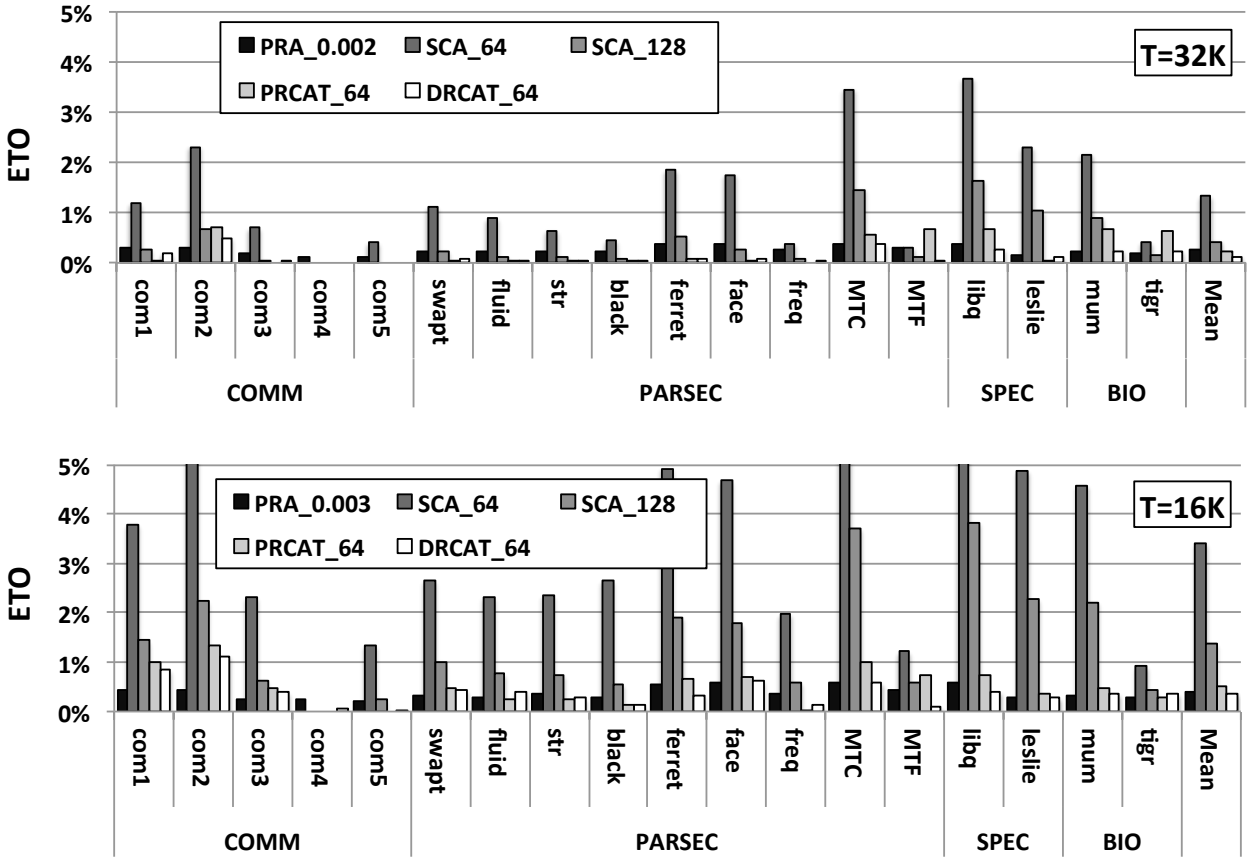


Figure 14: ETO resulting from refreshing vulnerable rows. DRCAT and PRCAT use 64 counters and up to 11 levels.

3.6 SENSITIVITY STUDY

3.6.1 Sensitivity to the Number of Counters and the Maximum CAT depth

Figure 15 shows CMRPO for DRCAT as the number of counters changes from 32 to 512 and the number of levels changes from 6 to 14, and compares results with those of SCA. From the figure, we note that increasing the number of CAT levels does not significantly impact CMRPO when the number of counters is relatively large. This is because, the static power consumed by the counters dominates the CMRPO, and hence, any improvement in the number of refreshed rows has minimal effect. Conversely, with a small number of counters, the energy for refreshing vulnerable rows is a large component of the CMRPO. Thus, having more levels in the tree saves refresh energy by targeting vulnerable row.

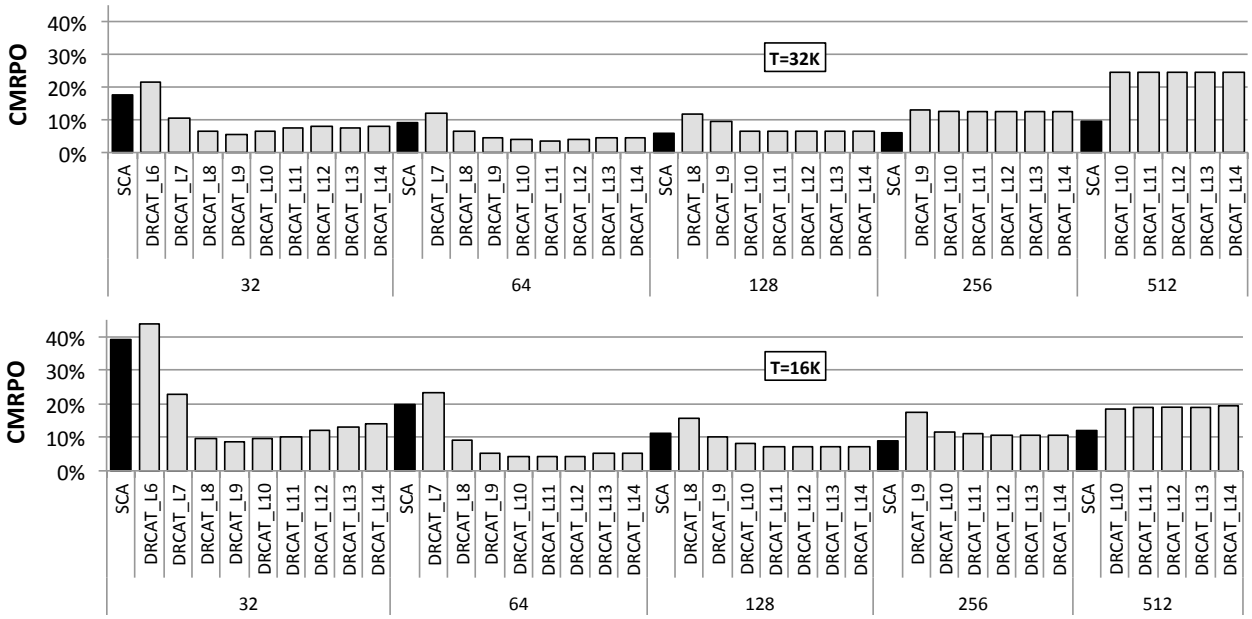


Figure 15: Crosstalk mitigation power overhead per bank for DRCAT using from 32 to 512 counters and different maximum CAT levels (6 to 14).

Due to the trade-off between static power and the power consumed to refresh vulnerable rows, the minimum CMRPO happens when DRCAT employs 64 counters and when SCA employs 128 counters for T=32K. Note that the refresh power of DRCAT_{64} with L7 is close to SCA_{64} since it only increases row resolution one more level beyond SCA_{64}. However, DRCAT_{64} incurs more static and dynamic power than SCA_{64}; hence, its CMRPO is larger. The same argument applies to explain why for fewer counters, CMRPO of SCA_{32} is smaller than that of DRCAT_{32}. When the threshold decreases from 32K to 16K, SCA will refresh victim rows more frequently and its CPRMO grows by 12% while the minimum CMRPO of DRCAT_{64} changes very little.

We studied the sensitivity of ETO to the number of counters and the tree depth (the results are not shown in this chapter). The key observation is that, for both refresh thresholds, when using at least 64 counters and $L \geq 9$, DRCAT incurs an ETO $< 1\%$. Results also show that ETO is inversely correlated to the refresh threshold. Another observation is that for a given fixed number of counters, increasing the tree depth does not necessarily reduce the number of refreshed victim rows; with a deeper tree, the number of rows associated with a certain counter will be reduced, but the number of rows associated with other counters

will increase. In other words, trying to be precise in one area of the memory may lead to a gross imprecision in another area of the memory, which creates a trade-off that leads to an optimum value for the maximum tree depth.

We conclude that for DRCAT, the optimal number of counters and the maximum CAT depth affect both the CMRPO and the ETO. For $T = 32K$ and $T = 16K$ and using between 32 and 128 counters, a maximum of $L = 11$ levels minimizes CMRPO and results in a low ETO. For CAT with more counters, the maximum CAT depth is inconsequential for CMRPO. In fact, using DRCAT leads to larger CMRPO than using SCA. We did the same analysis for PRCAT and our results show that CMRPO for PRCAT is about 4% and 7% for T=32K and T=16K with 10 and 11 CAT levels, respectively. Also it incurs very low performance overhead (<0.5%) for both thresholds.

3.6.2 Sensitivity to Mapping Policy and Number of Cores

To analyze the effect of address interleaving, we experiment with dual-core systems using two standard mapping policies of USIMM [Chatterjee et al., 2012]: (1) the 2-channel mapping policy (used in the experiments so far) and (2) a 4-channel mapping policy that maximizes memory access parallelism. Note that when keeping the size of each memory bank fixed, the 4-channel policy in USIMM quadruples the number of banks in the system. We also experiment with a quad-core system using the 2-channel and 4-channel mapping policies. The CMRPO of DRCAT, PRCAT and SCA are reported in Figure 16 for iso-area storage.

Figure 16 shows that, when using the 2-channel mapping policy, the CMRPO for quad-core systems is larger than the dual-core systems. This is because having more cores reduces the spatial locality in the L2 cache, thus generating more memory traffic and forcing more refreshes. SCA is affected more than the other schemes by the increased traffic because of the inability to accurately track the row accesses due to the uniform distribution of counters to rows. This effect is amplified when $T = 16K$ resulting in the CMRPO for SCA exceeding that of PRA for the quad-core system. In this case, DRCAT reduces the CMRPO in quad-core systems to 7%, which is an improvement over the 21% and 18% incurred in SCA and PRA, respectively. Figure 16 shows that for quad-core systems, the 4-channel policy reduces

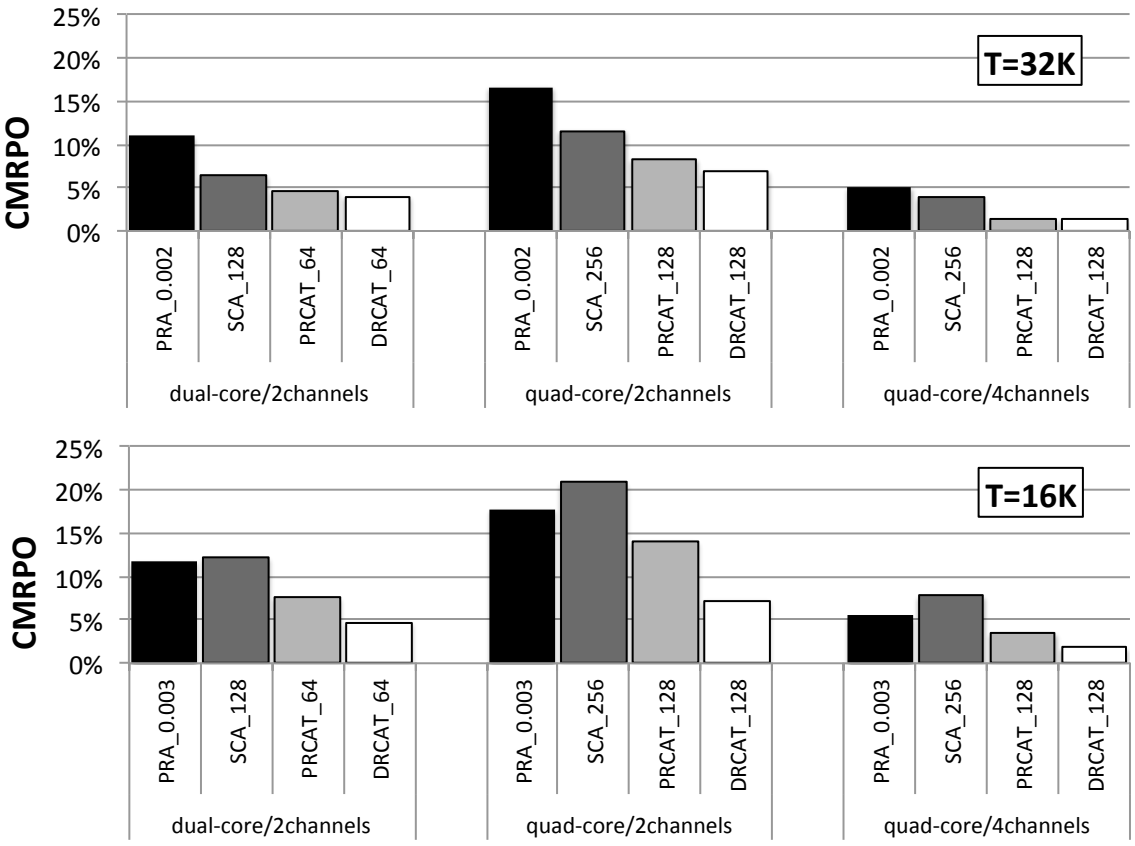


Figure 16: Effect of different mapping polices and number of cores on CMRPO (per bank). The banks in dual core and quad core systems include 64K and 128K rows, respectively.

CMRPO versus the 2-channel policy for schemes. This is expected since in the 4-channel policy, the number of banks increases from 16 to 64, thus decreasing refreshed rows.

Although we do not show the results for ETO in this section, we should note that ETO remains low for all schemes irrespective of the mapping policy or the number of cores. The largest ETO is incurred when the 2-channel policy is used with quad-cores and $T = 16K$. Specifically, in this case ETOs for $\text{PRA}_{0.003}$, SCA, PRCAT and DRCAT are 0.47%, 1.45%, 0.6%, 0.38% respectively. The relatively high ETO for SCA is due to the fact that the number of refreshed rows is relatively high.

3.6.3 Sensitivity to Refresh Thresholds

Scaling down DRAM technology exacerbates the crosstalk problem leading to a decrease in the refresh threshold [Kim et al., 2015]. This motivates the sensitivity analysis on different refresh thresholds presented in Figure 17, which shows the CMRPO for four refresh thresholds on a dual-core system with the 2-channel mapping policy. We used $\text{PRA}_{0.001}$, $\text{PRA}_{0.002}$, $\text{PRA}_{0.003}$ and $\text{PRA}_{0.005}$ for T = 64K, 32K, 16K and 8K, respectively to guarantee that the unsurvivability is better than 1.0E-4. The figure shows that, for thresholds 64K to 16K and dual core systems, DRCAT incurs CMRPO less than 5% which is an improvement over PRA's 12%. Also, it improves the CMRPO over PRCAT because the CAT is dynamically reconfigured rather than being periodically reset. Note that for T=8K, DRCAT and PRCAT need to double the number of counters to mitigate crosstalk, but still incur less than 10% CMRPO. With respect to ETO, all approaches incur very low overhead. Specifically, for $T = 8K$, the ETOs for PRA, SCA, PRCAT, DRCAT are 0.58%, 1.44%, 0.8%, and 0.48%, respectively. We conclude that CAT improves CMRPO relative to the other schemes for both current and future technologies.

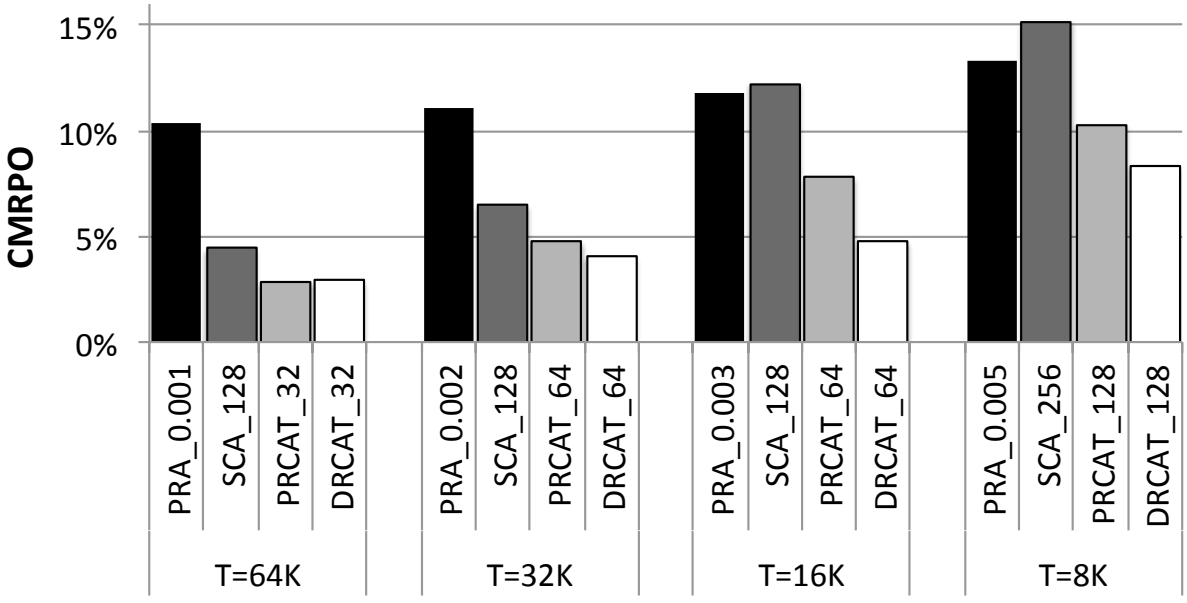


Figure 17: CMRPO for refresh thresholds T = 64K/32K/16K/8K.

3.6.4 Performance Under Malicious attacks

To evaluate the performance of the counter-based approaches under malicious attacks, we use 12 kernel attacks [Ghasempour et al., http://apt.cs.manchester.ac.uk/projects/ARMOR /RowHammer/ index.html] that randomly select few target rows (4 rows per bank and a total of 64 target rows for 16 banks with dual-core/2-channels configuration) and access the target rows more frequently than other rows in DRAM. We integrate the kernel attacks with regular access rows of memory-intensive workloads (which we call benign workloads). We select three attack modes *Heavy* (75% target rows + 25% benign access rows), *Medium* (50% target rows + 50% benign access rows) and *Light* (25% target rows + 75% benign access rows). Note that the distribution of target rows in the kernel attacks follows the Gaussian distribution. Figure 18 shows the average execution time overhead for the benign workloads for three refresh thresholds. As expected, more intensive attacks leads to higher ETO since it causes more refreshes. While the ETO for PRCAT and DRCAT is less than 0.9% and 0.6% for different attacks and refresh thresholds, the ETO of SCA grows to 4.5% for T=16K under heavy attacks. ETO for $T = 8K$ is lower than for $T = 16K$ because the number of counters is doubled. We conclude that when malicious attacks target specific rows in DRAM, CAT-based approaches are more efficient than SCA approaches at mitigating the attacks since they confine attacked rows to smaller groups of rows to be refreshed.

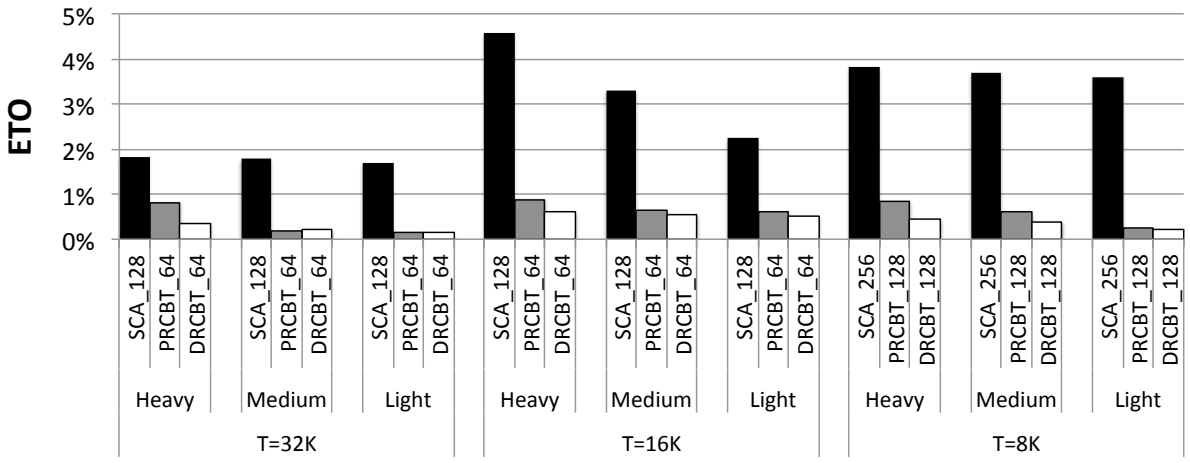


Figure 18: ETO for three kernel attack modes: Heavy (75% target rows + 25% benign access rows), Medium (50% target rows + 50% benign access rows) and Light (25% target rows + 75% benign access rows).

3.7 CONCLUSION

This chapter introduces the notion of a tree-based non-uniform row partitioning for detecting rows vulnerable to wordline crosstalk in memory banks. It proposes a low-cost implementation to maintain and access Counter-based Adaptive Trees that assign counters to rows non-uniformly and detects more precisely rows vulnerable to crosstalk. The results show that DRCAT outperforms the leading approaches for wordline crosstalk mitigation. Specifically, for dual-core systems and refresh threshold of $T = 16K$, DRCAT reduces the CMRPO to 4.5%, which is an improvement over the 12% and 22% incurred in deterministic and probabilistic approaches, respectively. Moreover, DRCAT incurs very low performance overhead ($< 0.5\%$). Hence, I conclude that dynamic row partitioning is an effective solution to detect rows vulnerable to crosstalk in DRAM. Clearly, this hardware solution avoids wordline crosstalk during normal execution and protects against malicious attacks.

4.0 LEVERAGING ECC TO MITIGATE READ DISTURBANCES, FALSE READS AND WRITE FAULTS IN STT-RAM

Designing reliable systems using scaled Spin-Transfer Torque Random Access Memory (STT-RAM) has become a significant challenge as the memory technology feature size is scaled down. The introduction of a more prominent read disturbance is a key contributor in this reliability challenge. However, techniques to address read disturbance are often considered in a vacuum that assumes that other concerns like transient read errors (false reads) and write faults do not occur.

This chapter studies several techniques that leverage ECC to mitigate persistent errors resulting from read disturbance and write faults of STT-RAM while still considering the impact of transient errors of false reads. In particular, It studies three policies to enable better-than-conservative read disturbance mitigation. The first policy, write after error (WAE), uses ECC to detect errors and write back data to clear persistent errors. The second policy, write after persistent error (WAP), filters out false reads by reading a second time when an error is detected leading to trade-off between write and read energy. The third policy, write after error threshold (WAT), leaves cells with incorrect data behind (up to a threshold) when the number of errors is less than the ECC capability.

4.1 MOTIVATION FOR INTRA-CELL DISTURBANCE MITIGATION

Figure 19 shows the probability of an error resulting from read disturbance in a 512-bit STT-RAM block as the number of read operation increases. The plot reveals that as the number of reads to a block increases, the probability of errors resulting from read disturbance

increases. Hence, mitigating destructive errors is a priority to improve the reliability of the system. The problem of read disturbance can be dealt with through writing back data after every read operation (WAR) [Sun et al.]. A write back mitigates disturbances, as the read data is correct due to the latent nature of errors. This approach makes the pessimistic assumption that reads are always destructive and mitigates read disturbances after every read operation. Clearly, this incurs an unnecessarily large overhead, as it is unlikely for every read operation to induce a disturbance. The goal of this chapter is to devise ECC-based solutions that detect and correct destructive read disturbances for different ranges of Raw Bit Error Rates (RBERs) whenever they occurs in the system instead of mitigating them after every read request.

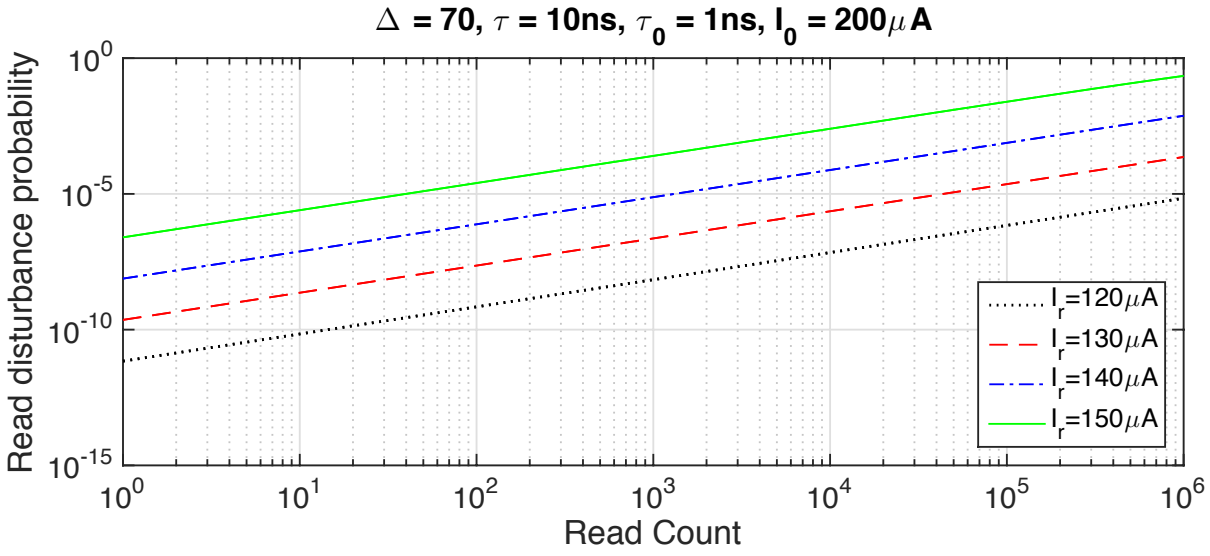


Figure 19: Probability of at least one error resulting from read disturbance in an STT-RAM cell relative to the number of reads [Sun et al.]. Parameters Δ, τ, τ_0, I_0 and I_r are explained in Section 2.2.2.

4.2 USING MARKOV CHAINS TO MODEL READ DISTURBANCE, FALSE READS AND WRITE FAULTS

To scrutinize the combined effect of persistent and transient errors on the UBER, we model read disturbances, write faults, and false reads using a *Markov Model* [Schiano et al.; Smyth, 1994]. A Markov chain can be described as a system that, at any time, is in one of a set of N states denoted by $S_1, \cdots, S_N$. Time is divided into steps and at any step, t, the system can switch to another state with a given probability. The probability to go from state S_i to state S_j does not depend on the specific step t and is denoted c_{ij}. To obey standard stochastic constraints, $c_{ij} \geq 0$ and $\sum_{j=1}^{N} c_{ij} = 1$. A Markov chain may have one or more absorbing states. By definition, the state S_i is absorbing when $c_{ii} = 1$ (and hence $c_{ij} = 0$ for all $j \neq i$). Any absorbing chain can be specified by a canonical form [Grinstead and Snell, 2012; Turin and Sondhi, 1993] from which the expected number of steps (state transitions) to absorption can be estimated [Kitchin].

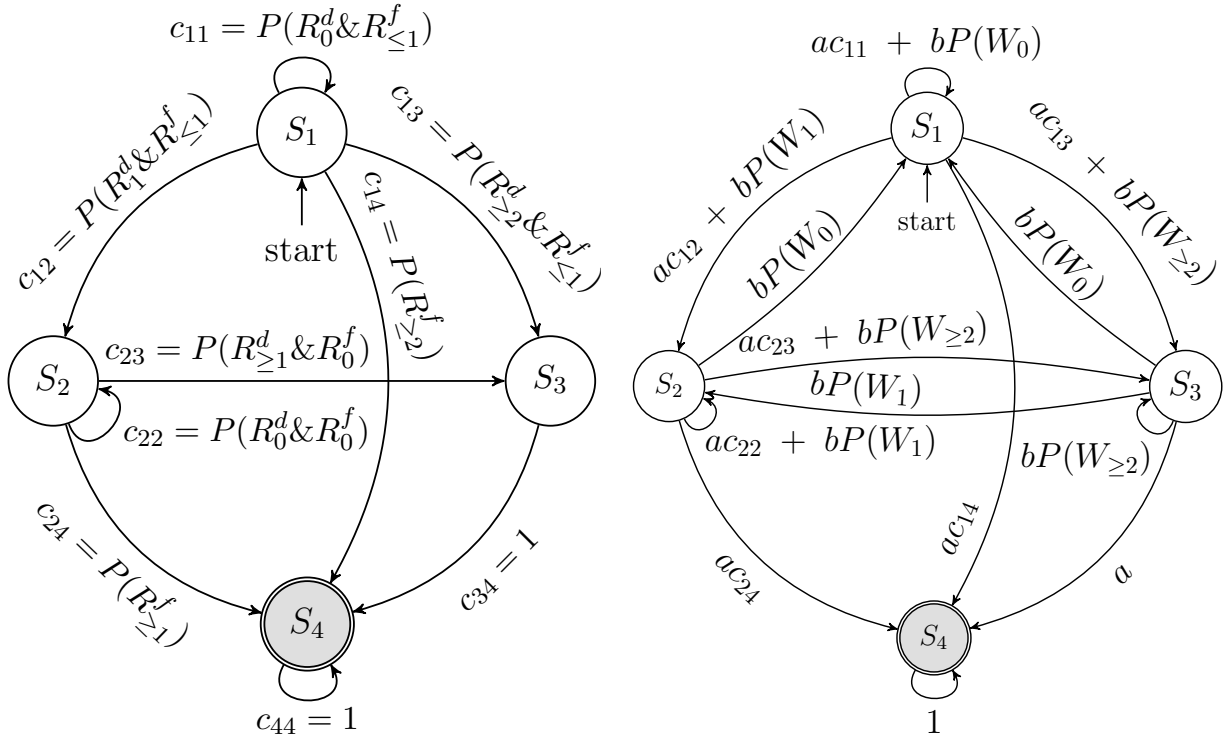
Markov analysis is a suitable option for modeling systems with read disturbance and calculating the probability of failure for such systems. To make the idea more concrete, we describe a Markov model of the process of repeatedly reading an m-bit data block protected by a single error correcting code (ECC_1) in the presence of both read disturbances (with probability p_d) and false reads (with probability p_f). Specifically, repeatedly reading the data with no intervening user write operations can be described by the four-state Markov model shown in Figure 20(a), where each transition represents a read operation. In this model, the probability of disturbance is fixed for each read and the cumulative effect of read disturbance is captured by the different Markov states. Specifically, states S_1, S_2, or S_3 represent the states where zero, one, or at least two cells in the data block contain wrong data, respectively. Initially, the system is in S_1 and eventually, the system will be absorbed in the "failure to correctly read" state, S_4. Assuming that n is the size of the block including the ECC bite ($n > m$), the probability that x cells are disturbed during the read operation can be computed as $P(R_x^d) = \binom{n}{x} p_d^x (1-p_d)^{n-x}$ [Grinstead and Snell, 2012]. Similarly, the probability of y false reads occurring during the read operation is $P(R_y^f) = \binom{n}{y} p_f^y (1-p_f)^{n-y}$. In the figure, $P(R_{\geq x}^d)$ denotes the probability that at least x cells are disturbed during the read

operation and $P(R_x^f \& R_y^d)$ denotes the probability of x false reads and y read disturbances.

After a read operation, the data block remains in state S_1 as long as no cell is disturbed and at most one read error occurs, which can be corrected by ECC_1. If one cell is disturbed during the read, the state of the block transitions from S_1 to S_2. In the next read operation, the error is detected and corrected by the ECC but the stored value is not corrected (assuming no provision is made to deal with read disturbance). Provided that no new cells are disturbed in consecutive read operations, the Markov process remains in S_2. If more than one cell is disturbed in the next read operations, the state of the block changes from S_2 to S_3. In this case, the process definitely (with probability 1) moves to S_4 by the next read because the number of disturbed cells is more than the ECC capability and the errors cannot be corrected. Note that state S_3 is needed because a read operation that disturbs a cell retrieves the value stored in that cell correctly but every subsequent read suffers from an error due to the disturbed cell. Note also that a false read does not have a cumulative effect and only influences the present status of the Markov process. That is, if the number of false reads lies in the range of the ECC capability, errors are corrected and do not appear in consequent read operations.

Using standard Markov analysis, we can calculate the expected number of transitions before absorption. Accordingly, the inverse of the calculated value over m gives the UBER. As an example, consider the case of a 64-bit block with the probabilities of read disturbance and false reads being $p_d = p_f = 10^{-6}$ and the codeword length being $n = 71$. Starting at S_1, the expected number of transitions before absorption into S_4 is 21127, which leads to $UBER = \frac{1}{21127 \times 64} = 7.39 \times 10^{-07}$.

So far, we have assumed that every transition is due to a read operation. Normally, however, a data block is subject to both read and write operations and we can use a Markov process to model a system in which $a\%$ of the operations are reads and $b = (100 - a)\%$ are writes. A write operation will remove the effect of any previous read disturbance, but may introduce a "write fault". The model of Figure 20(a) can be extended to Figure 20(b) to include the effect of user write operations and the probability of write faults. In that extension, the meanings of the states S_1, ... ,S_4 are unchanged with the understanding that a cell may contain a wrong value due to a write fault as well as a read disturbance.



(a) Read disturbances and false reads.

(b) Read disturbances, false reads and write faults.

Figure 20: Modeling the state of a data block protected by ECC_1.

New transitions (edges) are added to the model to represent user write operations, with the understanding that writing new data into the block clears any previously faulty cells (returns the state to S_1), unless a write fault occurs (with some probability, p_w), in which case the process transitions to either S_2 or S_3, depending on the number of write faults. Although modeling write operations complicates the Markov process, the reliability of the process can be evaluated using the same technique. Assuming that $a = 99.9\%, b = 0.1\%$, and $p_w = p_f = p_d = 10^{-6}$ in the previous example of using ECC_1, we can compute the number of transitions before failure to be 120421, which leads to $UBER = \frac{1}{120421 \times 64} = 1.29 \times 10^{-07}$. This shows that although errors may happen during writes, the system is more reliable because write operations store correct data into cells that were affected by read disturbances.

4.3 REVISITING WRITE BACK AFTER USER READ

One possible solution for mitigating read disturbance in STT-RAM is for the system to induce a write back of the data block after every user read (WAR) [Sun et al.]. In Figure 21, we show the Markov model for WAR when used in conjunction with ECC_1. For simplicity of the presentation, we assume no user write operations ($a = 100\%$ and $b = 0\%$) noting that it can be easily added as was described in the previous section. In the figure, black edges represent user read operations and blue dashed edges represent system write back operations.

We define S_1 as the initial state of the data block. A user read transitions the state of the block to S_5 (the failure to "read" state) if more than one false reads occur because this exceeds the correction capabilities of ECC_1. If, however, at most one false read occurs, the block transitions to S_2 and a system write back occurs. Depending on the errors during the write back, three scenarios may occur. (1) If no write faults occur, the state of the block returns to S_1. (2) If one fault occurs during the write back, the block transitions to S_3 which indicates that one cell contains the wrong data. Being in S_3, the next read operation will either fail (if one or more false reads occur) or the wrong cell will be detected/corrected and the block will be written back transitioning the block to state S_1 (through S_2). Finally, (3) If more than one fault occurs during the write back, the block will have two cells with

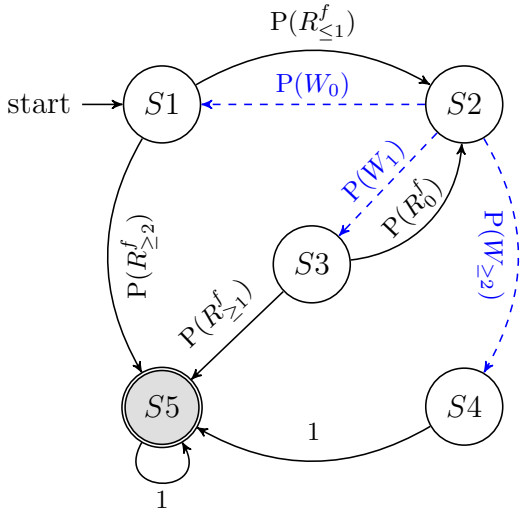


Figure 21: Modeling write back after read (WAR).

Table 4: Comparison of WAR and ECC_1^{64} in terms of UBER for different ranges of p_w, p_f and p_d.

Parameter	Scheme	$RBER(p_w)$			
		10^{-4}	10^{-5}	10^{-6}	10^{-7}
$p_w = 10p_f$	WAR	$2.23\ 10^{-05}$	$2.34\ 10^{-07}$	$2.35\ 10^{-09}$	$2.35\ 10^{-11}$
$= 10p_d$	ECC_1^{64}	$1.54\ 10^{-05}$	$1.63\ 10^{-07}$	$1.64\ 10^{-09}$	$1.64\ 10^{-11}$
$p_w = p_f = p_d$	WAR	$7.74\ 10^{-07}$	$7.81\ 10^{-09}$	$7.82\ 10^{-11}$	$7.82\ 10^{-13}$
	ECC_1^{64}	$1.52\ 10^{-06}$	$1.56\ 10^{-08}$	$1.56\ 10^{-10}$	$1.56\ 10^{-12}$
$p_w = 10^{-5}p_f$	WAR	$1.93\ 10^{-07}$	$1.94\ 10^{-09}$	$1.94\ 10^{-11}$	$1.94\ 10^{-13}$
$= 10^{-5}p_d$	ECC_1^{64}	$8.58\ 10^{-07}$	$8.79\ 10^{-09}$	$8.81\ 10^{-11}$	$8.81\ 10^{-13}$

incorrect data (state S_4) and the next read operation will not be able to correct the errors, causing a read failure.

Using the Markov models, we compare in Table 4 the UBER when repeatedly reading a 64-bit data block protected by ECC_1 with and without WAR. From the results, we observe that when the write bit error rate increases relative to the read bit error rate (including false reads and read disturbances), the UBER exponentially grows and WAR becomes less reliable than ECC_1. This means that although WAR mitigates the read disturbance for high p_w, it generates more errors during the write back process. Accordingly, WAR decreases reliability when cells encounter a high write error rate compared to the read error rate.

Another drawback of WAR is its energy overhead, since it requires an expensive write operation after every read (in STT-RAM, writing a cell consumes at least four times the energy of reading it [Meza et al., 2012; Mishra et al., 2011]). Moreover, the write backs consume a large portion of the memory bandwidth (again, in STT-RAM a write is much slower than a read). Hence, although the system write back is not on the critical path of a user read operation, it may delay subsequent read operations because of memory bandwidth saturation. These observations motivate the solutions described in the next section.

4.4 ON-DEMAND WRITE BACK POLICIES

Depending on the dominant error type in STT-RAM, we describe three policies to mitigate the effects of read disturbance, write faults and false reads: (1) Write back After any Error detection (WAE), (2) Write back After Persistent error detection (WAP), and (3) Write back After errors reach a Threshold (WAT). The main advantage of these policies is that they avoid the unnecessary write backs based on the observation that it is unlikely that every user read operation will induce a disturbance, thereby they significantly reduce both the energy overhead and memory bandwidth overhead caused by the unnecessary system writes in WAR.

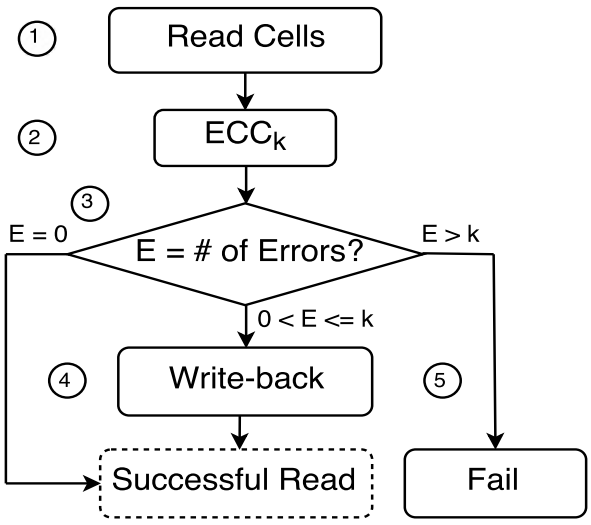
The key idea of WAE is to write back data only when ECC detects errors. It leverages ECC_k during the user read operation to detect and correct up to k errors of the data block. When errors are corrected, the corrected data is written back. The flowchart in Figure 22(a) pictorially depicts the process of reading a data block using WAE. It is composed of five steps:

- The controller first reads the data block from memory ①.
- It then applies ECC_k ② to the block.
- If no errors are detected, the reading succeeds ③.
- Else, if the number of detected errors lies within the range of the ECC capability, the data block is written back ④.
- Else, the reading fails because the data cannot be correctly retrieved ⑤.

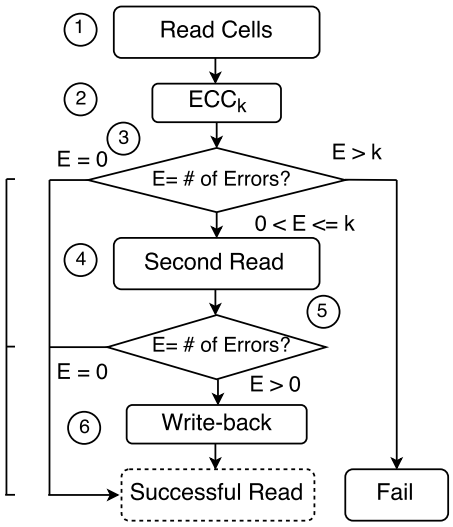
WAE writes back the data block after an error is detected, even if this error is due to a false read rather than a read disturbance. The key idea of WAP is to filter out false reads when ECC detects errors by reading the data block again (a second read). Specifically, when errors are detected during the user read operation, WAP corrects the read data by ECC provided that the number of errors is within the ECC capability. Then, the policy performs a second read (termed a system read since it is not requested by the user) and compares the read value with the corrected data block. A write back of the corrected block is performed if the comparison reveals a discrepancy.

Figure 22(b) pictorially depicts the process of reading a data block using WAP. Specifically,

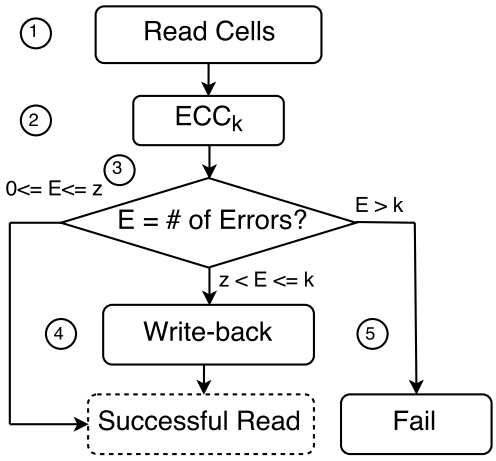
- The first three steps of WAP are the same as WAE, except that a copy of the corrected data is kept if errors are detected and are within the ECC capability.
- The fourth step reads data again ④ if ECC detects correctable errors during the previous read. Then, the data blocks of the previous (corrected) and current reads are compared.
- If the two blocks are identical, the reading succeeds ⑤.



(a) Write after any error detection.



(b) Second read after error detection.



(c) Leave z errors behind, $z < k$.

Figure 22: The flowcharts of on-demand write-back policies.

- Else, the correct data is written back ⑥. Note that cells that may be newly disturbed in the second read will be corrected due to the write back operation.

The choice between WAE and WAP depends on the read and write energy cost and the proportion of the user read to the user write. When read disturbance is dominant, WAP utilizes additional reads increasing its energy overhead relative to WAE. On the other hand, when read disturbance is not dominant, WAP reduces the number of system write backs thus reducing the energy overhead and the memory bandwidth relative to WAE. This advantage increases when the cost of write operations increases relative to the cost of read operations.

Clearly, WAP reduces the number of write backs over WAE by introducing more reads. It is possible, however, to reduce the number of write backs without the additional reads if some detected faults are explicitly left behind. This, of course, can only be done if an ECC_k is used with $k > 1$. For example, if an ECC_2 is used, then it is possible to avoid the write back operation when one error is detected. This will avoid unnecessary write backs when false reads occur but do not correct read disturbances unless a second error is detected. We call this policy of leaving cells with incorrect data behind up to a threshold which is less than the ECC capability "WAT".

The main rationale behind the design of WAT is that when the false read error rate is high and the dominant errors are not destructive, most errors are transient and hence cells do not have to be refreshed after detecting errors. Accordingly, WAT leaves read disturbances and write faults behind and ignores false reads up to a threshold, z, that is within the ECC capability (i.e. $z < k$). Therefore, the only difference between WAT and WAR is that it postpones the system write back until the number of errors in the data block reaches z. This reduces the energy overhead compared to the other approaches. Figure 22(c) depicts WAT which includes the following steps:

- The first two steps are the same as the previous policies.
- In the third step ③, three different cases can occur: 1) if the number of errors is less than or equal to z, the policy leaves cells with incorrect data behind and the reading succeeds, 2) if the number of errors is greater than z and less or equal to k, the policy writes back the data, 3) if the number of errors is more than k, the reading fails.

Table 5: The interpretation of states for WAE and WAP.

State	WAE	WAP
S_1	No error/disturbance	No error/disturbance
S_2	One error	One error
S_3	One undetected disturbance	At least one error in the second read
S_4	At least two undetected disturbances	One undetected disturbance
S_5	At least two errors	At least one undetected disturbance
S_6	-	At least two undetected disturbances
S_7	-	At least two errors

For different policies, the ECC capability plays a significant role in determining the conditions for a failed read operation. The more powerful ECC schemes allow the policies to increase the expected number of reads before failure, which leads to smaller UBER, and fewer write back and read operations. Considering the ECC with k error correction capability, we constructed Markov models of WAE, WAP and WAT. They are composed of $k+4$, $k+6$ and $k+4$ states, respectively. In the next section, we describe, in some details, the Markov models for the three policies, WAE, WAP and WAT, when k is 1, 1 and 2, respectively.

4.5 RELIABILITY ANALYSIS OF THE DIFFERENT SCHEMES VIA MARKOV MODELS

We use Markov models to determine the reliability of the policies described in Section 4.4 and quantify, on average, the number of write backs and second reads for each policy. For simplicity, we assume in this section only user read operations (no user write) and no write error during the write back process for the different policies. Similar to what was explained for Figure 20(b), the user write operations and the effect of p_w can be added to each model and will be considered in the experimental results.

Table 6: The interpretation of states for WAT.

State	WAT
S_1	At most one error
S_2	Two errors
S_3	One disturbance (either detected or undetected)
S_4	At most one error and two undetected disturbances
S_5	At most one error and at least three undetected disturbances
S_6	At least three errors
S_7	-

4.5.1 Write back After Error detection (WAE)

First, we discuss the Markov model of WAE that leverages ECC_1. Figure 23(a) illustrates that model where each edge is labeled by the probability of transition between states, which is based on the type and number of errors occurring during the user read operations or system write back operations. Note that a cell affected by a read disturbance during a read will only cause an error during the next read. The interpretation of states of different policies is illustrated in Tables 5 and 6. As shown in Figure 23(a), initially, before any read operation, the block is in the error-free state, S_1. After a read operation, the block stays in the same state, S_1, in case of no read error or read disturbance, but transitions to another state according to the number and nature of the error. Specifically,

- If a false read occurs in one cell, it is detected by ECC_1 and the system writes back the block. This is shown by the transition to S_2 (a temporary state) with probability $P(R_1^f)$ followed by a transition back to S_1 with probability 1. Note that we made the assumption that there is no write errors during write backs ($p_w = 0$).
- If false reads occur in more than one cell, ECC_1 cannot correct the errors and the process transitions to S_5, the Failure state.
- If no false reads occur but a read disturbance occurs in one cell, the process transitions to S_3 (denoting one cell affected by a read disturbance). Then, in the following read operation either (1) no read errors occur and the ECC_1 will detect the disturbance error and force

a write back after correcting the error (the process transition to S_2 and then to S_1 with probability 1), or (2) some false reads occur, thus exceeding the correction capability of ECC_1 and transitioning the process to the failure state.

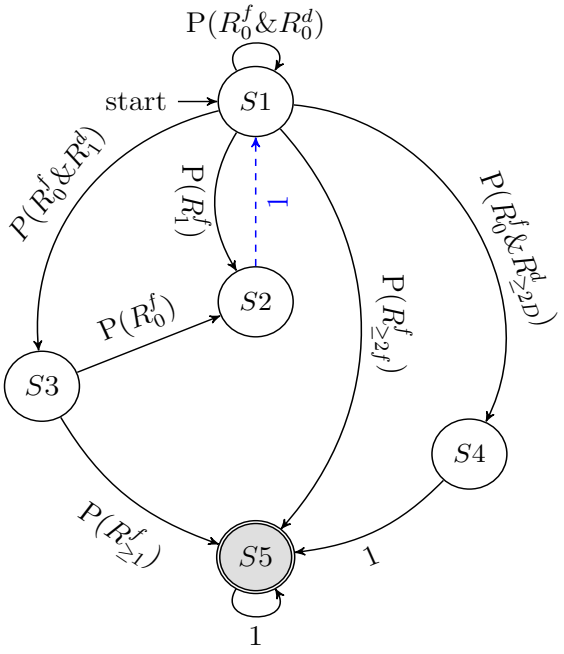
- If no false reads occur but read disturbances occur in more than one cell, the process transitions to S_4 (denoting more than one cell affected by read disturbance). In this case, ECC_1 will not be able to correct the disturbance errors in the next read operation, taking the process to the Failing state.

The standard Markov analysis can be used to compute the number of times every edge in the model is traversed before absorption. This allows us to compute the number of system write operations ($S_2 \rightarrow S_1$ transitions) relative to the number of user read operations (the other transitions). Specifically, the number of time an edge ($S_i \rightarrow S_j$) is traversed can be computed as the product of the number of times state S_i is visited before absorption and the probability of the transition $S_i \rightarrow S_j$. From a probabilistic point of view, there are two points to note in Figure 23(a): (1) the number of write backs depends on two very low probability events ($S_1 \rightarrow S_2$) and ($S_1 \rightarrow S_3 \rightarrow S_2$), and (2) the probability of no false read and no read disturbance, $\mathrm{P}(R_0^f \& R_0^d)$, is higher than other events. Therefore, the Markov process mostly remains in S_1 and avoids unnecessary writes.

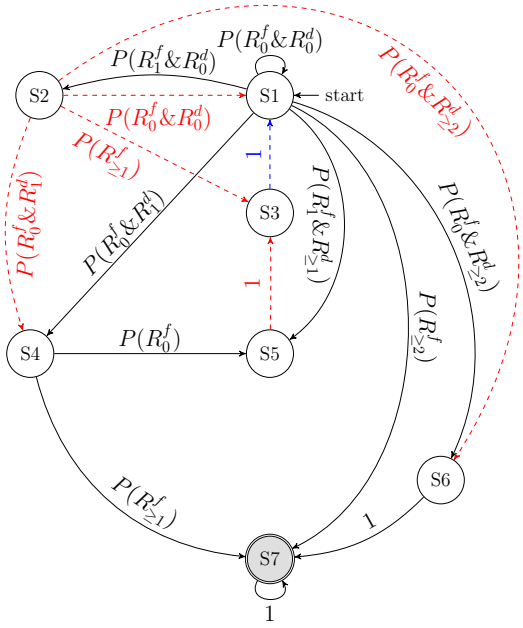
4.5.2 Write back After Persistent error detection (WAP)

Similar to what was explained for WAE, we model WAP for ECC_1 using the Markov model shown in Figure 23(b). Specifically, starting from the initial state S_1, a user read operation can cause the following state transitions:

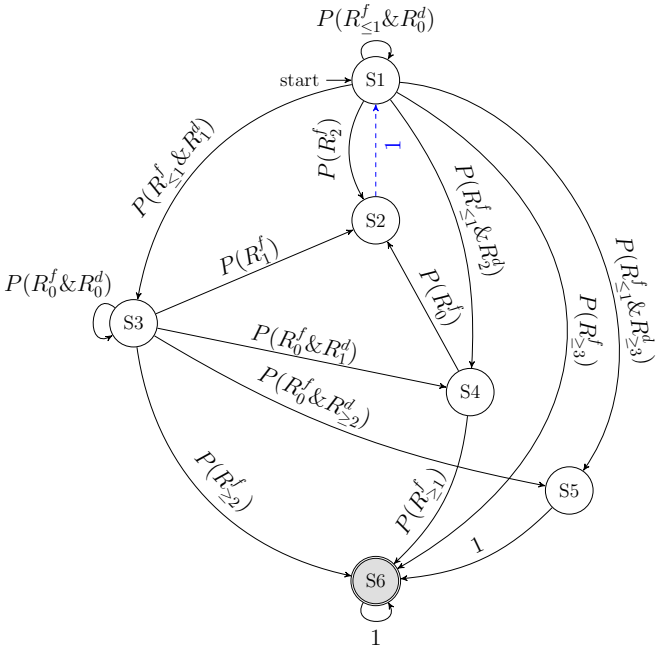
- If no error or disturbance occur, the process stays in S_1.
- If the number of false reads exceeds the ECC capability (larger than 1), the process transition to S_7 (the failure state).
- If the number of read disturbances exceeds the ECC capability (larger than 1) as long as no false read occurs, the process moves to S_6 (at least two disturbed cells). The next user read ($S_6 \rightarrow S_7$) will cause a transition to the failure state, S_7.



(a) WAE used with ECC_1.



(b) WAP used with ECC_1.



(c) WAT used with ECC_2.

Figure 23: The Markov models for on-demand write-back policies. The red and blue dashed links indicate the second read (system read) and write back (system write) operations, respectively.

- If only one cell is disturbed but no false read occurs, the disturbance is not detected and the process moves to S_4. The next read operation will then detect the error and transition the process to the failure state if a false read occurs ($S_4 \rightarrow S_7$) since the capability of ECC will be exceeded. If, however, no false read occurs in the next read operation ($S_4 \rightarrow S_5$), the latent read disturbance will be detected and the read data block is corrected and kept. Then, the system initiates a second read (system read) and compares two data blocks. This second read will confirm that the detected error is due to the disturbed cell ($S_5 \rightarrow S_3$) and will write back ($S_3 \rightarrow S_1$) the corrected data.

- If at least one cell is disturbed and exactly one false read occurs, the process moves to S_5 where ECC detects the error and corrects the data and the system triggers a second read and a comparison between previous and current read data ($S_5 \rightarrow S_3$). The comparison reveals the occurrence of at least one persistent error; therefore the process writes back the corrected data block.

- If exactly one false read occurs, ECC detects the error and corrects the data and the process moves to S_2. In this case, a second (system) read is initiated which may lead to four different transitions depending on what happens during this second read:

  (1) No cell is disturbed and no false read occurs. In this case, the process moves to the initial state ($S_2 \rightarrow S_1$).

  (2) No cell is disturbed but at least one false read occurs (the comparison between previous and current read data reveals the occurrence of the error(s)). In this case, the system read takes the process to S_3 ($S_2 \rightarrow S_3$) and the corrected data will be written back ($S_3 \rightarrow S_1$).

  (3) One cell is disturbed during the system read and no false read occurs. In this case, the process moves to S_4 ($S_2 \rightarrow S_4$).

  (4) If the number of read disturbances exceeds the ECC capability (larger than 1) as long as no false read occurs, the process moves to S_6 (at least two disturbed cells). The next user read ($S_6 \rightarrow S_7$) will cause a transition to the failure state.

4.5.3 Write back After error Threshold (WAT)

Since leaving at least one cell with incorrect data or one false read needs an ECC that detects two errors, ECC_2 is the smallest ECC code which can be considered for WAT. Figure 23(c) shows the Markov model for WAT based on ECC_2. It can be described as follows:

- As long as the number of errors detected during a read operation is less than two, the Markov process ignores false reads and leaves cells with incorrect data behind. Considering only the number of read disturbances occurring during the read operation, the process transitions to a state that keeps track of the number of disturbances. Specifically, S_1, S_3, S_4 and S_5 reflect zero, one, two and more than two disturbed cells, respectively.

- If two false reads are detected but no read disturbance occurs, the process is transitioned to S_2 and a write back of the corrected data returns the system to S_1.

- From a state that reflects x disturbed cells, a read operation with more than $2 - x$ false reads causes a transition to the failure state, S_6.

- From S_3 (one cell is already disturbed), a read operation with one false read will detect two errors, correct the errors and write back the block ($S_3 \rightarrow S_2 \rightarrow S_1$), clearing the disturbance.

- From S_4 (two cells are already disturbed), a read operation with no false read will detect two errors, correct the errors and write back the block ($S_4 \rightarrow S_2 \rightarrow S_1$). However, any false read in the read operation cannot be corrected (ECC_2 cannot correct more than 2 errors) and will take the system to failure (S_6).

As mentioned earlier, user write operations as well as the effect of write errors can be incorporated in the above three Markov models, and the UBER as well as the overhead (in terms of system write back and second read operations) of the policies can be computed. In the evaluation section, we report the results of our analysis of the models that incorporates user write operations and user read operations for write faults, false reads, and read disturbances.

4.5.4 Accounting for miscorrections and undetected errors

Usually, ECC code that can correct k errors can detect $k+1$ errors but can produce the wrong data if more than $k+1$ error occurs. For example, if errors change one codeword to another codeword, the errors are not detected. Moreover, if the errors change a code word, x, to a non-code word, y, and the Hamming distance between x and y is larger than the Hamming distance between y and another code word, z, then the errors will be miscorrected to z. Our failing states combines detected failures and miscorrections/non-detections. To more precisely differentiate the two cases, we can decompose the failing state, S_f, into two absorbing states, miscorrection/non-detection, S_{mc} and detected-failure, S_{df}. This only nominally increases the complexity of the model (one additional state) and will allow differentiation between S_{mc} and S_{df}. The expected number of transitions before absorption is the same in both cases. Moreover, our observation based on experimental results shows that the probability of absorption to S_{mc} is much smaller than to S_{df} (e.g., S_{ms} is 10^6 times smaller than absorption to S_{df} for the model shown in Figure 21).

4.5.5 Markov models for other memory technologies

According to the type of errors in other technologies, Markov chains can be designed to model the combined effect of persistent and transient errors including read disturbance and estimate the reliability of the systems. In STT-RAM, since the disturbance does not affect neighboring cells, the Markov models are simpler than other technologies, such as PCM, where disturbance errors (write disturbance) can affect, not only the accessed cell, but also neighboring cells.

4.6 EVALUATION

4.6.1 Baseline

We evaluate the reliability and energy consumption of the error mitigation policies assuming 64-bit data blocks for a range of read disturbance, write error and false read error rates. Since ranges of these three types of error rates in STT-RAM depend on various circuit and

system parameters such as circuit configuration, read and write currents, pulse-widths, etc., we evaluate both single MTJ [Kang et al., 2013; Zhang et al., 2012] and dual-MTJ [Zhang et al.] STT-RAM configurations with a range of corresponding read and write currents and pulse widths. Note that we do a sweep of parameters but can not report the large volume of results. Instead, we report results that are representative of the sweep and reflect cases that arise from specific practical technologies. We include cases that span all the possible relative orders of the values of p_f, p_d, and p_w.

For a single MTJ STT-RAM configuration, when the reading current varies from $24.5\mu A$ to $41.5\mu A$, p_f varies from 10^{-4} to 4×10^{-5} while p_d varies from 10^{-10} to 10^{-4}. Also, the write current was set to $I_w = 56.1\mu A$ [Kang et al., 2013] and with a write pulse of 10ns and thermal stability $\Delta = 45$ the write error is approximately $p_w = 10^{-10}$. We also varied the write pulse time from 10ns to 4ns with a corresponding pulse width $\frac{\tau}{\tau_0}$ of 100% to 40%, leading to p_w that varies from 10^{-10} to 10^{-4}. For the dual-MTJ STT-RAM configuration, when the reading current varies from $50\mu A$ to $70\mu A$, the false read bit error rate improves by more than two orders of magnitude while degrading the read disturbance error rate by an order of magnitude. Also, with a write current $I_w = 98.5\mu A$, the write bit error rate is reported as $p_w = 1.2 \times 10^{-7}$ [Zhang et al.]. We conducted experimental results data points in these ranges shown in Tables 7, 8, and 9.

Table 7: Bit error rates of different types of errors in terms of corresponding currents for single MTJ STT-RAM.

I_R (μA)	24.5	27.5	30.8	33.2	36.6	41.5
p_d	$1 \cdot 10^{-10}$	$1 \cdot 10^{-9}$	$1 \cdot 10^{-8}$	$1 \cdot 10^{-7}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-5}$
p_f	$1 \cdot 10^{-4}$	$9 \cdot 10^{-5}$	$8 \cdot 10^{-5}$	$7 \cdot 10^{-5}$	$6 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
$log(p_f/p_d)$	6	4.954	3.903	2.845	1.778	0.698

Table 8: Bit error rates of different types of errors in terms of corresponding currents for dual-MTJ STT-RAM.

I_R (μA)	50.0	53.6	57.2	60.8	64.4	70.0
p_d	$1 \cdot 10^{-9}$	$1 \cdot 10^{-8}$	$5 \cdot 10^{-8}$	$1 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	$5 \cdot 10^{-6}$
p_f	$1 \cdot 10^{-6}$	$7 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$7 \cdot 10^{-8}$	$3 \cdot 10^{-8}$	$1 \cdot 10^{-8}$
$log(p_f/p_d)$	3	1.845	0.602	-0.154	-1.221	-2.698

Table 9: Write bit error rate by changing the write pulse width.

τ (ns)	10	9	8	7	6	5	4
$\frac{\tau}{\tau_0}$	100%	90%	80%	70%	60%	50%	40%
p_w	$1 \cdot 10^{-10}$	$1 \cdot 10^{-9}$	$1 \cdot 10^{-8}$	$1 \cdot 10^{-7}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-4}$

4.6.2 Uncorrectable Bit Error Rate

Figure 24 shows UBER for the proposed error mitigation policies against different values for the ratio of false read to read disturbance for standard, single MTJ STT-RAM when using single bit error correction and a $p_w \approx 10^{-10}$. The x-axis data points correspond to the different values for I_R reported in Table 7 for different read currents. When the access pattern is equal (50%) user reads and writes all approaches achieve similar UBER level as the user write requests compensate for the effect of cumulative read errors from read disturbance. For a 1000 to 1 read to write ratio, the reliability of approaches does not change significantly, as I_R increases to improve p_f, the resulting higher p_d increases the probability of read disturb errors. Thus, ECC_1 alone cannot correct the read disturbances effectively and it becomes less reliable than other approaches that include some policy for writing back.

When using two-bits error correction, we can also now consider WAT. Figure 25 shows similar UBER for different policies based on ECC_2 with all policies performing similarly for equal (50%) user reads and writes and with a similar trend where ECC_2 scales poorly when user reads dominate and read disturbance becomes high. Similarly, since WAT leaves cells with incorrect data, it follows a similar trend to ECC_2 but extends the useful range through an additional data point making it useful for moderate rates of read disturbance where ECC alone is not as successful.

We conclude that when the user read to write ratio increases, if the read disturbance error rate is significant and comparable to other bit error rates from other factors, the system reliability due to error mitigation policy varies significantly. Thus, in the remainder of the evaluation we conduct experimental results on a "worst-case" ratio of 1000 user reads to each user write (a=99.9% vs. b=0.1%).

Table 10 shows the UBER versus different RBERs for the ratios of p_f and p_d specified

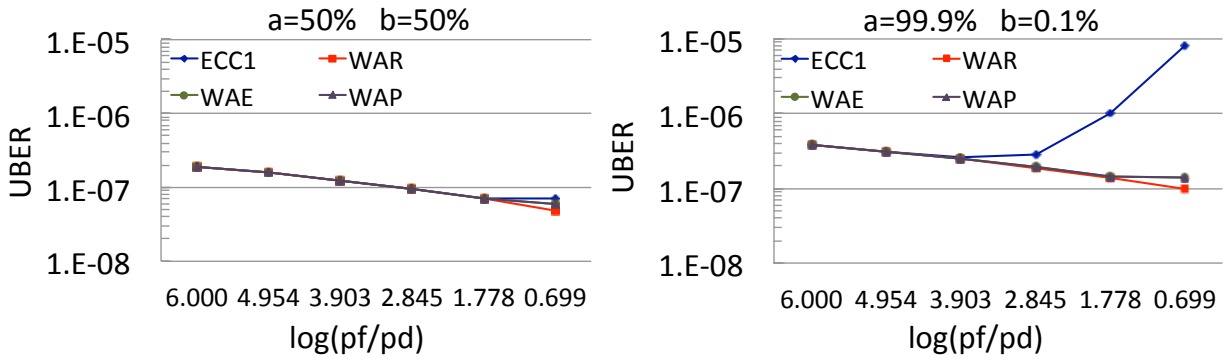


Figure 24: Uncorrectable bit error rate vs. row bit error rates under two ratios of read to write operations for single MTJ STT-RAM. All approaches leverage ECC_1.

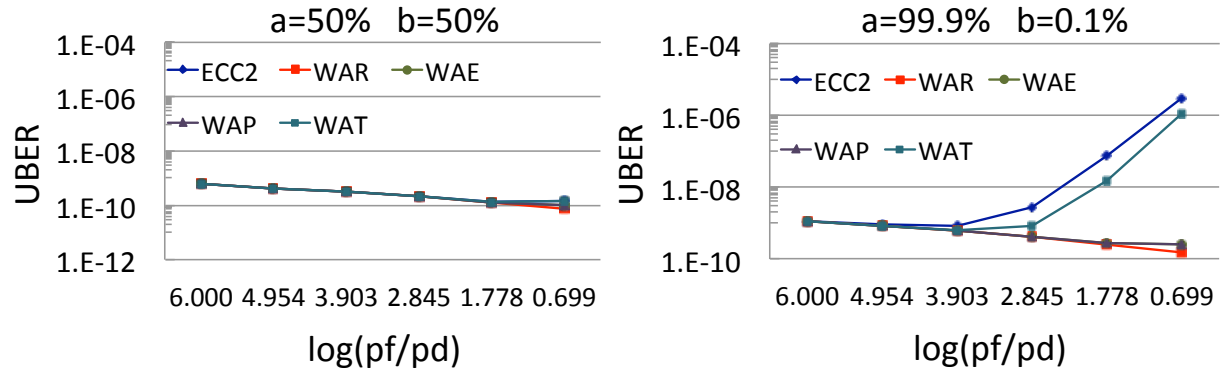


Figure 25: Uncorrectable bit error rate vs. different row bit error rates under ratios of read to write operations for single MTJ STT-RAM. All approaches leverage ECC_2.

in Table 8 for a dual-MTJ STT-RAM. Since dual-MTJ for STT-RAM reduce p_f by more than two orders of magnitude and increase p_d and p_w by more than one and three orders of magnitude, respectively, the relationships are different than in the single MTJ case. When p_f is the dominant error rate in the system, WAR, WAE and WAP are more reliable than ECC_k, where k is the number of bits that can be corrected, and WAT. When the rate of p_d grows in comparison to p_f and p_w, WAR is more reliable than other approaches due to its consistent write back to eliminate read disturbance. However, WAR incurs a high energy overhead and consumes significant additional memory bandwidth for this reliability benefit. Moreover, writes are expensive operations and significant effort has been applied to reduce the impact of writing into an MTJ. One method is to reduce the write pulse width τ and tolerate a higher p_w. If $p_w \leq \{p_f, p_d\}$ then WAR also becomes both expensive for energy

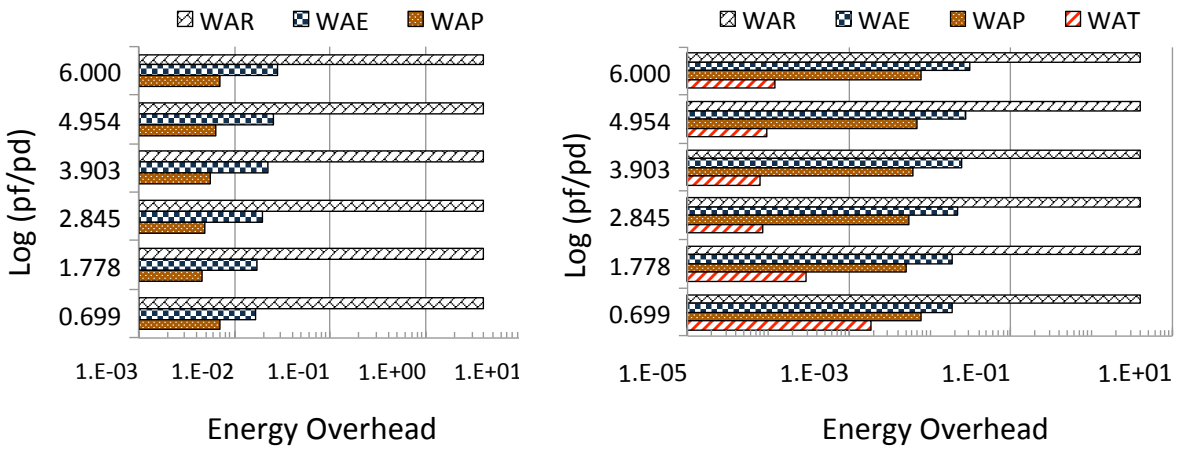
Table 10: The comparison of different policies across different RBERs for STT-RAM. All Parameters are from dual-MTJ STT-RAM.

$log(p_f/p_d)$		3	1.845	0.602	-0.154	-1.221	-2.698
k=1	ECC_k	$4.45\ 10^{-11}$	$2.80\ 10^{-11}$	$1.15\ 10^{-11}$	$1.22\ 10^{-11}$	$1.65\ 10^{-10}$	$1.51\ 10^{-08}$
	WAR	$4.44\ 10^{-11}$	$2.39\ 10^{-11}$	$3.72\ 10^{-12}$	$1.34\ 10^{-12}$	$8.43\ 10^{-13}$	$6.38\ 10^{-13}$
	WAE	$3.59\ 10^{-11}$	$1.82\ 10^{-11}$	$2.35\ 10^{-12}$	$1.05\ 10^{-12}$	$8.93\ 10^{-12}$	$7.89\ 10^{-10}$
	WAP	$3.59\ 10^{-11}$	$1.82\ 10^{-11}$	$2.35\ 10^{-12}$	$1.05\ 10^{-12}$	$8.93\ 10^{-12}$	$7.89\ 10^{-10}$
k=2	ECC_k	$1.31\ 10^{-14}$	$5.22\ 10^{-13}$	$4.52\ 10^{-12}$	$1.22\ 10^{-11}$	$8.72\ 10^{-10}$	$3.53\ 10^{-07}$
	WAR	$1.70\ 10^{-15}$	$6.72\ 10^{-16}$	$3.83\ 10^{-17}$	$6.18\ 10^{-18}$	$2.07\ 10^{-18}$	$5.57\ 10^{-19}$
	WAE	$1.19\ 10^{-15}$	$4.25\ 10^{-16}$	$2.03\ 10^{-17}$	$6.18\ 10^{-18}$	$1.77\ 10^{-16}$	$1.49\ 10^{-13}$
	WAP	$1.19\ 10^{-15}$	$4.25\ 10^{-16}$	$2.03\ 10^{-17}$	$4.44\ 10^{-18}$	$1.77\ 10^{-16}$	$1.49\ 10^{-13}$
	WAT	$1.22\ 10^{-14}$	$4.88\ 10^{-13}$	$3.57\ 10^{-12}$	$5.05\ 10^{-12}$	$5.16\ 10^{-11}$	$1.19\ 10^{-09}$

and potentially less reliable than other methods. In the following section, we explore the trade-off between the system reliability and the energy overhead for different approaches.

4.6.3 Energy Overhead Evaluation

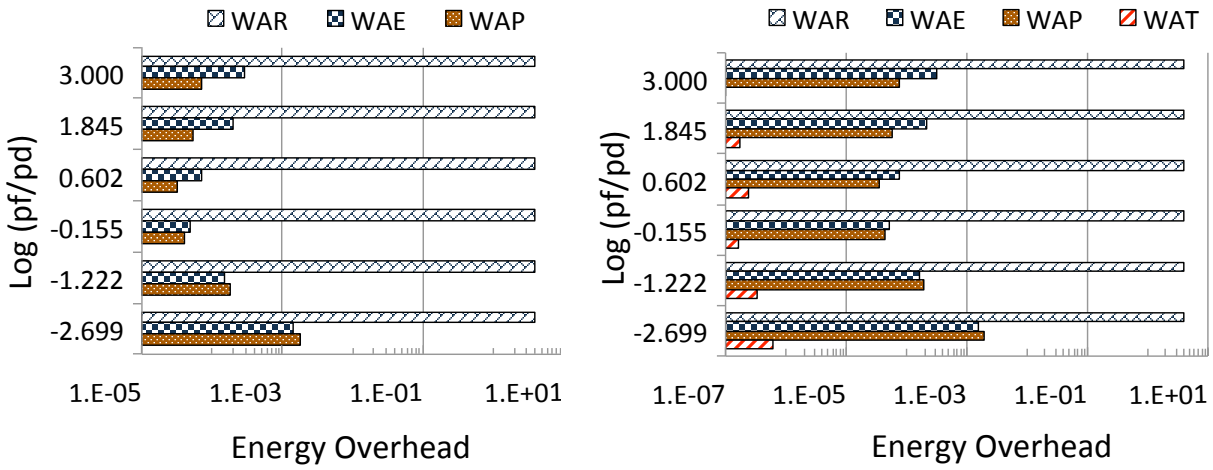
We define the read energy overhead as the energy consumed for all the system operations (write backs and second reads) relative to the energy for all the user operations. That is, the average increase in the energy needed for a user operation. Assuming that every write operation consumes 4 times the energy for a read operation [Meza et al., 2012; Mishra et al., 2011], the energy overhead for WAR is 400% because each user read is followed by a system write back. Figure 26 breaks down the average energy overhead of different error mitigation policies for a single MTJ STT-RAM. Compared to WAR (400%), the average energy overhead of WAE and WAP is less than 2% and 0.5%, respectively. As long as the dominant error in the system is false read (as is the case for the single MTJ configuration), WAP reduces energy overhead over WAE. When available (e.g., with ECC_2) WAT achieves



(a) All approaches utilize ECC_1.

(b) All approaches utilize ECC_2.

Figure 26: Comparing the average energy overhead of different approaches. Parameters are for single MTJ STT-RAM.



(a) All approaches utilize ECC_1.

(b) All approaches utilize ECC_2.

Figure 27: Comparing the average energy overhead of different approaches. Parameters are for dual-MTJ STT-RAM.

a significant energy savings particularly in cases where p_f dominates [Figure 26(a)].

For a dual-MTJ STT-RAM, results in Figure 27 shows that average energy overhead of WAE, WAP, and WAT is very low, reaching 0.0397%, 0.0392%, and 0.00002%, respectively. Compared to single MTJ STT-RAM, dual-MTJ STT-RAM reduces average energy overhead for these policies as p_f is reduced by more than two orders of magnitude. In fact, p_f no longer dominates as p_d and p_w increase by more than one and three orders of magnitude, respectively

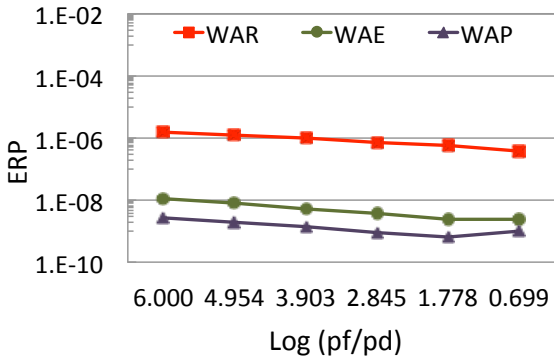
and become higher than p_f. Note that when p_d and/or p_w is greater than p_f (negative points of the vertical axis in Figure 27), the dominant errors are persistent. Thus, WAP uses an additional read to detect persistent errors which becomes less valuable, so its energy overhead becomes greater than WAE, which immediately writes back on any error. Accordingly, the double read by WAP and or the write back by WAE after error detection can prevent the unnecessary system operations while retaining an acceptable UBER level. WAT, however, can dramatically reduce the average energy overhead compared to other approaches as it can tolerate some cells with incorrect data before writing back while still achieving a satisfactory UBER for certain values of I_R.

We conclude that writing back a data block after every read operation incurs a large overhead and other approaches dramatically reduce this energy overhead while achieving a similar or acceptable UBER level.

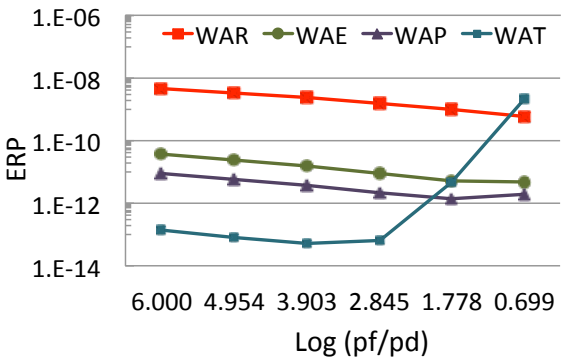
4.6.4 Energy Reliability Product

Product metrics, such as Energy-delay product [Horowitz et al.; Sato and Funaki] are common to evaluate trade-offs between two metrics. To evaluate the trade-off between energy overhead and reliability, we use the ERP metric, which we define as the product of the energy overhead *times* UBER. Thus, we utilize the ERP metric to evaluate the efficiency of different approaches in a similar fashion. While WAR can have significant ramifications on delay because every access incurs a write operation, as WAE, WAP, and WAT do not write back data blocks frequently, and additional reads from WAP are also infrequent, their delay is negligible compared to WAR and small compared to ECC alone. Figure 28 shows that WAE and WAP for single MTJ STT-RAM improve the energy reliability product by more than two orders of magnitude compared to WAR for different ratios of RBERs. Furthermore, when p_f is greater than p_d and p_w, WAT dramatically improves the energy reliability product by more than two orders of magnitude versus WAE and WAP and more than five orders of magnitude compared to WAR. When P_d dominates, the advantage of WAT further increases.

In dual-MTJ STT-RAM, the decrease in false read error rate versus single MTJ STT-

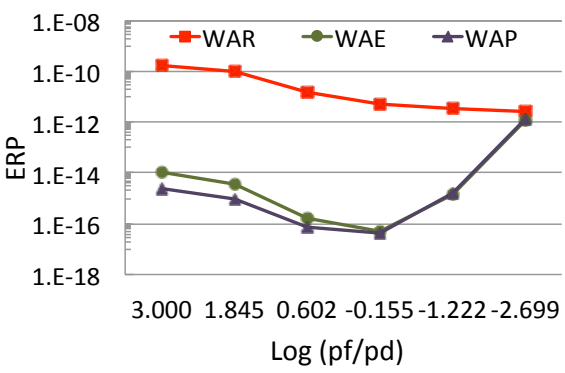


(a) All approaches utilize ECC_1.

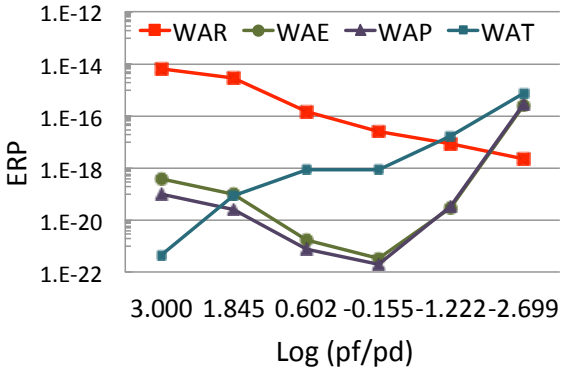


(b) All approaches utilize ECC_2.

Figure 28: Energy reliability product of different approaches. All parameters are for the single MTJ STT-RAM.



(a) All approaches utilize ECC_1.



(b) All approaches utilize ECC_2.

Figure 29: Energy reliability product of different approaches. All parameters are for dual-MTJ STT-RAM.

RAM results in WAE and WAP improving the energy reliability product over WAR by roughly the same degree, as shown in Figure 29. When p_d overtakes p_f (negative points of the horizontal axis), WAE and WAP obtain the same energy reliability product due to similar UBERs and energy overheads. However, when p_d is high, WAE still has an advantage over WAP which requires the second read to filter out false reads. Unsurprisingly, as p_d increases, WAT does not eliminate read disturbances effectively and the cumulative effect of read disturbance causes the energy reliability product to quickly drop below WAE and WAP as shown in Figure 29(b).

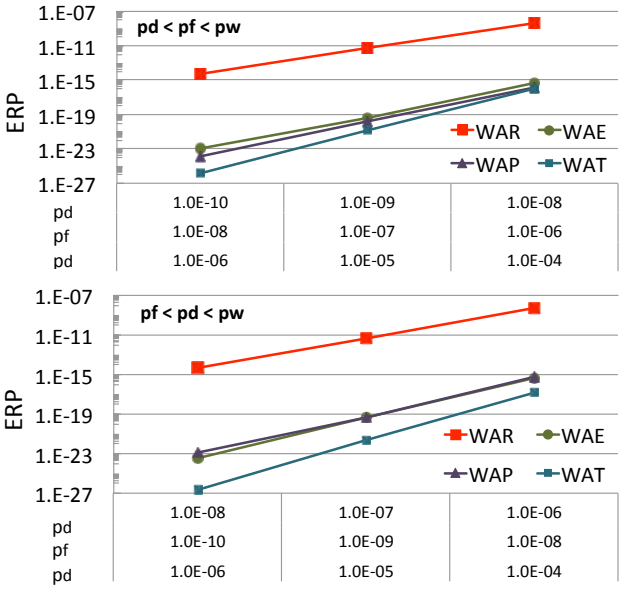
We conclude that although these approaches tend to deliver a better system failure rate than other approaches, they can incur a high energy overhead. Our evaluation based on energy reliability product shows that WAT and WAR achieve the best and worst performance, respectively for a single MTJ STT-RAM in which the false read error rate is the dominant in the system. In dual-MTJ STT-RAM where p_d and potentially p_w are more important, WAE and WAP obtain best performance since they enable better read disturbance mitigation.

4.7 SENSITIVITY ANALYSIS

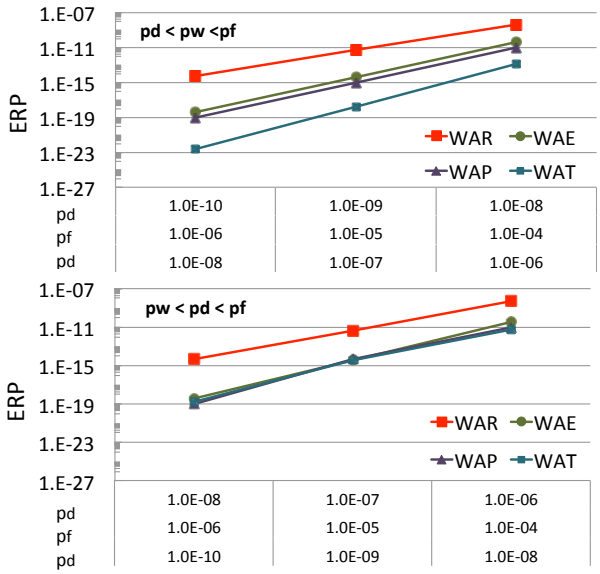
In Section 4.6.2, we showed that a high incidence of user writes can eliminate the destructive effects of read disturbances. However, heavily read data (e.g., 1000 reads or more to one write) does yield cumulative error effects due to read disturbance. Furthermore, results in Section 4.6.4 show that the energy reliability product for different mitigation approaches depends on ratios of raw bit error rates p_f, p_d, and p_w due to circuit parameters in STT-RAM and that the type of dominant error rate plays a significant role in the overall system performance for different error mitigation policies. We now explore the energy reliability product of these different approaches in a systematic way so that p_d, p_f, and p_w can change in three different modes: high, medium and low. Figure 30 plots the energy reliability product of different approaches against three different modes of p_d, p_f, and p_w. For example, Figure 30(a) shows ERP of different approaches when p_w has the highest bit error rate which varies between 10^{-6} and 10^{-4} and p_d and p_f have medium error (varying between 10^{-8} and 10^{-6}) and low error (varying between 10^{-10} and 10^{-8}). Similarly, Figure 30(b) and 30(c) show the cases where p_f and p_d dominate the error rate, respectively. As before, we characterize the energy reliability product versus raw bit error rates for the user read write ratio ($a = 99\%$ and $b = 0.1\%$) and make the following observations:

- When p_w is the highest bit error rate in the system as shown in Figure 30(a) [This scenario is relevant to an STT-RAM system with a reduced write pulse τ to save write energy]:
  - WAT achieves the best performance against other approaches because it tends to write back less and avoids the high p_w. However, if the system does experience enough cells with incorrect data from p_w and p_d it will write back to eliminate the accumulated errors.

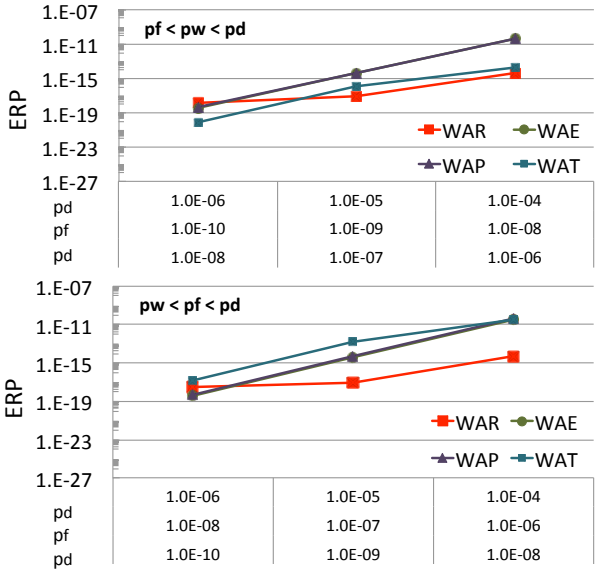
- While WAR typically performs poorly in terms of energy, but well in terms of reliability, this is an example where WAR can hurt reliability and energy making it the worst performing approach. Furthermore, its negative impact on latency due to the high



(a) p_w is the highest bit error rate.



(b) p_f is the highest bit error rate.



(c) p_d is the highest bit error rate.

Figure 30: Energy reliability product of the different policies for six different scenarios.

memory write bandwidth makes this a poor choice for a system with a high p_w.

- When p_f is the highest bit error rate in the system as shown in Figure 30(b) [This scenario is relevant to a "standard" single MTJ STT-RAM system with a standard write pulse]:
  - When p_d is the lowest bit error rate amongst the three, WAT achieves the best ERP, which is reduced on average by more than three orders of magnitude compared with WAE and WAP and also eight orders of magnitude smaller than WAR.
  - When $p_d \approx p_f$, WAE, WAP, and WAT have the same ERP all of which are better than WAR. The destructive effects of p_d degrade the energy reliability advantage of WAT relative to the case $p_f >> p_d$.
- When p_d is the highest bit error rate in the system as shown in Figure 30(c) [This scenario is relevant to the dual-MTJ STT-RAM approach which increases p_d and reduces p_f and/or when I_R is increased to reduce p_f but increases p_d]:
  - WAR has the best ERP among all approaches primarily because writing back the data block consistently has a significant reliability advantage that outweighs the energy savings in this scenario.
  - As long as p_w is the lowest bit error rate in the system, WAT obtains the worst ERP because the combination of p_f and p_d have a higher incidence of multiple errors occurring after a single error was left behind.
  - If p_f is the lowest bit error rate in the system ($p_d > p_w > p_f$), the performance of WAE and WAP exponentially degrade.

4.8 CONCLUSION

Spin-Transfer Torque Random Access Memory (STT-RAM) is one of the leading candidates in emerging memory technologies. Unfortunately, the relatively unreliable reads of the STT-RAM due to read disturbances degrades system reliability and precludes the integration of STT-RAM into the memory stack. In this chapter, we studied three approaches to mitigate read disturbances for STT-RAM compared to the conservative approach of writing back

after every read. Further, these approaches are designed to improve overall memory system reliability by addressing read disturbance, write faults and false read errors together.

These techniques leverage a single ECC to cover all three different types of faults/errors. In particular, we considered schemes to write back blocks after any error is detected (WAE), after a persistent error (due to read disturbance or write fault) is detected (WAP), or after multiple errors are detected (WAT). Further, we provided a description of a Markov modeling approach that evaluates all three types of errors and generates a single reliability of the system in terms of uncorrectable bit error rate. Moreover, we described energy reliability product metric to be able to quantitative evaluate the trade-off between system energy and reliability. Our study concludes the following:

1. WAT, the design that potentially leaves behind some cells with incorrect data due to faults as long as the number of errors is less than or equal to an error threshold, achieves the best energy reliability trade off when the false read error rate or the write bit error rate is dominant in the system.
2. WAE, the design that writes back data after detecting any error achieves acceptable reliability and energy levels, as long as the read disturbance is not dominant.
3. WAP, the design that reads data again after detecting an error to distinguish between transient and persistent errors has an energy and reliability that is similar to that of WAE, as long as the read disturbance is not dominant.
4. WAR does not have a significant reliability advantage over the other policies when the read disturbance is not dominant in the system. Moreover, WAR has the highest energy and memory bandwidth overheads among all the policies.

In summary, we showed qualitatively that WAE, WAP and WAT provide dramatic improvement in energy consumption and memory bandwidth overhead (due to additional write-backs) while eliminating the effects of read disturbance and retaining near WAR reliability.

5.0 INTEGRATING MULTI-TIERED COMPRESSION WITH COSET CODING FOR PCM TO MITIGATE WRITE DISTURBANCES

Phase change memory (PCM) has recently emerged as a promising technology to meet the fast growing demand for large capacity memory in computer systems, replacing or augmenting DRAM that is impeded by physical limitations. To achieve high memory capacity, the wordline and bitline distances contract in super dense PCM which easily increases sneak heat among cells along the wordline and bitline intensifying write disturbances. While the sneak heat for resetting cells slowly decays vertically along the bitline, it diminishes fast horizontally along the wordline. Thus, the likelihood of incidence of write disturbance along the bitline is more than that along the wordline. When the write disturbance occurs in the neighboring wordlines of the active wordline, they have to be read&written to eliminate write disturbances which sacrifices system performance. It is possible to thwart the bitline write disturbances by not reducing inter-cell spacing along the bitlines. This solution however may only be suitable in Multi-Level Cell (MLC) PCM since it compensates cutting memory capacity at the expense of higher write energy and wordline disturbance error rates. The naive solution to reduce write energy in MLC PCM or likelihood of incidence of write disturbance in SLC PCM is to use coset coding whose cost function optimizes objectives such as write energy, disturbance errors and etc. Coset coding is a mapping function that maps each dataword to multiple coset candidates and the coset candidate with the minimum cost is selected to be written in the memory. The coset candidates are indexed by auxiliary bits in the encoder that sacrifice capacity. One solution for salvaging memory capacity is to store auxiliary bits in the compressed cacheline assuming they do not exceed the number of reclaimed bits by the compression.

The goal of this chapter is to propose a generic approach that integrates a new multi-

tiered compression (Section 5.1) with coset coding to optimize the cost function. To this end, we characterize the realistic workloads and explore them for the multi-tiered compression that does not disturb in-place similarity [1] for application to optimize write energy in MLC PCM and likelihood of incidence of write disturbance errors in SLC PCM. The new compression technique compresses more than 94% of the memory lines and provides enough room within the cacheline to store the auxiliary data using defined coset encoding applied at data block granularities lower than the typical cache line size. Finally, we show how to make trade-off among reliability, performance, energy and endurance. This work provides the ground work needed to tackle the third research question listed in Section 1.2. Section 5.1 introduces the new multi-tiered compression. Sections 5.2 and 5.3 tackle write disturbances in MLC and SLC PCM, respectively.

5.1 MULTI-TIERED COMPRESSION (MTC)

The so called "in memory compression" algorithms are based on the similarity, measured by the Hamming distance, among neighboring data elements [Kim et al., a; Pekhimenko et al., 2012; Seol et al.; Yang et al., 2000; Zhang et al., 2000]. Some compression algorithms such as BDI [Pekhimenko et al., 2012] measure the Hamming distance in order to exploit the dynamic range of values, which is common in integer and pointer array types. BDI then encodes a block of data as a single base value, followed by a set of differences relative to that base. In contrast, some compression algorithms such as Bit Plane compression [Kim et al., a] start with a smart data transformation on a set of bits corresponding to the same bit position within each word in a data array (bit plane) to improve the compressibility of data while keeping the encoding complexity comparable to existing compressors. These bit plane transformations are then combined with the existing lightweight compressors (such as BDI) to turn the improved compressibility into real bit savings. Unfortunately, these existing compressors change the bits sufficiently to harm *in-place similarity* sufficiently to defeat much of the savings from differential write [Seyedzadeh et al., 2018].

[1] *In-place similarity* is the similarity of the old data to the corresponding new data in the memory line.

We apply different existing schemes for compressing 512-bit memory lines to SPEC2006 and PARSEC workloads. The results show that FPC+BDI compression [Alameldeen and Wood, 2004; Pekhimenko et al., 2012] only compresses 30% of the memory lines. In contrast, the recently proposed Coverage-Oriented Compression (COC) [Kim et al., b] highly compresses cache lines by utilizing 28 different variable length compressors. Unfortunately, the variable length encoding used in COC disturbs the biased bit patterns in a memory line. However, by inspecting each 64-bit memory word, we found that there is a significant similarity across and within most significant bytes of most words. We take advantage of this feature to propose a new Multi-Tiered Compression technique that does not disturb in-place similarity. To this end, we divide each cacheline into eight 64-bit words and then explore similarity in the most significant bits $b(i,j)$ of each word w_i where $0 \leq i \leq 7$ and $56 \leq j \leq 63$. Our exploration reveals similarity of bits within and across words as follows:

- The last 'k' significant bits of each word have the same values: $b(i, 64-k) = b(i,j)$ where $0 \leq i \leq 7$, $64-k \leq j \leq 63$ and $2 \leq k \leq 8$. In this case, '$(k-1) \times 8$' bits are reclaimed and eight most significant bits from the same bit locations across different words are selected as the compression encoding bits. Note that this specific compression is called Word Level Compression (WLC_k). Given k=8 and k=6, $\text{WLC}_{k=8}$ and $\text{WLC}_{k=6}$ reclaim 56 and 40 bits as shown in Figures 31(a) and 31(b).

- All bits corresponding to the same bit position across words have the same values as shown in Figure 31(c): $b(0,j) = b(i,j)$ where $1 \leq i \leq 7$ and $56 \leq j \leq 63$. Similar to previous case, bits $b(0,j)$, where $56 \leq j \leq 63$, are considered as the compression encoding bits and 56 bits are reclaimed. Note that this specific compression is called Cross Word Level Compression (CWC) .

Figure 32 compares the percentage of compressible cachelines. While WLC_7 and WLC_8 equally compress on average 54% of the cachelines, WLC_4, WLC_5 and WLC_6 equally compress 90% of the cachelines. To constitute the Multi-Tiered Compression (MTC) that compresses many cachelines and reclaims many bits, WLC_6, WLC_8 and CWC are selected. The main reason for selecting two versions of WLC is that WLC_8 reclaims 56 bits and WLC_6 by reclaiming 40 bits compresses a high percentage of cachelines. Figure 32 shows that

MTC compresses more than 94% of the cachelines while not disturbing in-place similarity. Note that the number of reclaimed bits by MTC ranges from 40 to 56. In Sections 5.2 and 5.3, we will use reclaimed bits by WLC_6 and MTC to store auxiliary bits of coset coding, respectively.

	b63	b62	b61	b60	b59	b58	b57	b56	...	b0
W0	0	0	0	0	0	0	0	0	...	x
W1	0	0	0	0	0	0	0	0	...	x
W2	1	1	1	1	1	1	1	1	...	x
W3	0	0	0	0	0	0	0	0	...	x
W4	1	1	1	1	1	1	1	1	...	x
W5	0	0	0	0	0	0	0	0	...	x
W6	0	0	0	0	0	0	0	0	...	x
W7	1	1	1	1	1	1	1	1	...	x

(a) Similarity of each bit position within MSBs*.

	b63	b62	b61	b60	b59	b58	b57	b56	...	b0
W0	0	0	0	0	0	0	x	x	...	x
W1	0	0	0	0	0	0	x	x	...	x
W2	1	1	1	1	1	1	x	x	...	x
W3	0	0	0	0	0	0	x	x	...	x
W4	1	1	1	1	1	1	x	x	...	x
W5	0	0	0	0	0	0	x	x	...	x
W6	0	0	0	0	0	0	x	x	...	x
W7	1	1	1	1	1	1	x	x	...	x

(b) Similarity of leading six bits of each MSB*.

	b63	b62	b61	b60	b59	b58	b57	b56	...	b0
W0	1	1	0	0	0	0	0	0	...	x
W1	1	1	0	0	0	0	0	0	...	x
W2	1	1	0	0	0	0	0	0	...	x
W3	1	1	0	0	0	0	0	0	...	x
W4	1	1	0	0	0	0	0	0	...	x
W5	1	1	0	0	0	0	0	0	...	x
W6	1	1	0	0	0	0	0	0	...	x
W7	1	1	0	0	0	0	0	0	...	x

(c) Similarity across MSBs*.

Figure 31: Multi-Tiered Compression (MTC). The red, blue and black values represent compression bits, reclaimed bits and data bits, respectively. *Most significant *byte*.

5.2 REDUCING WRITE DISTURBANCE IN MLC PCM

When the inter-cell spacing along the bitline remains untouched in super dense PCM, the likelihood of incidence of bitline write disturbances is negligible but it sacrifices memory capacity. MLC PCM delivers higher memory capacity at the expense of higher programming energy and higher wordline write disturbance error rates. In this section, first we integrate the word level compression, WLC_6, with coset coding in order to minimize write energy in

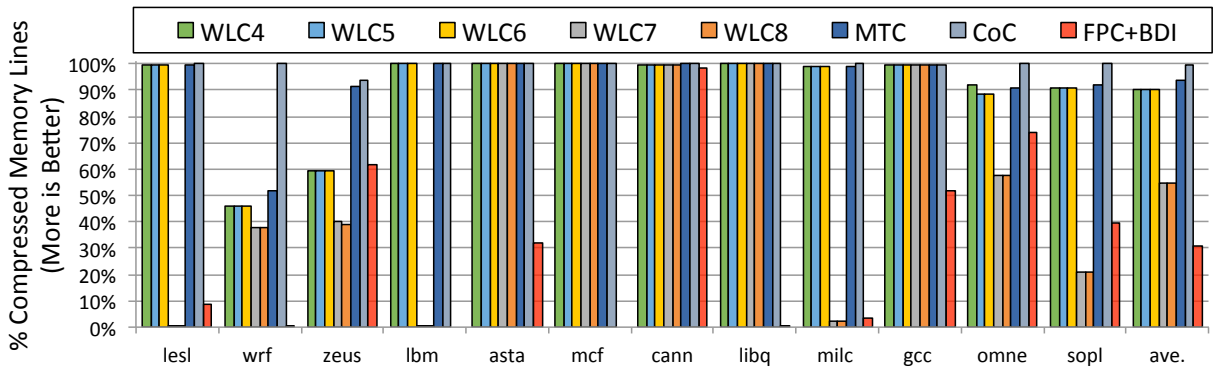


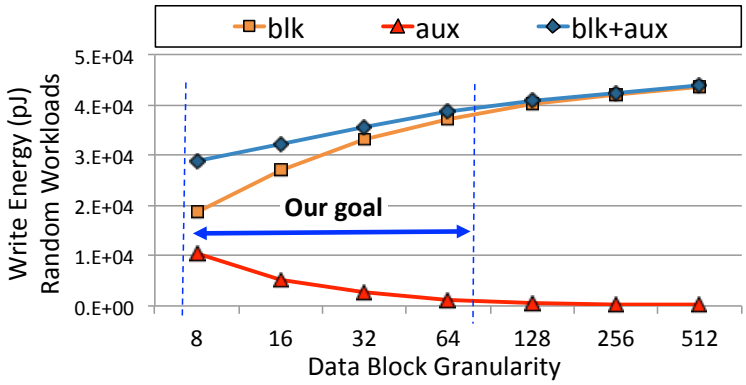
Figure 32: Comparison of the percentage of compressed memory lines by WLC, MTC, COC [Kim et al., b] and FPC+BDI [Pekhimenko et al., 2012].

MLC PCM. Then, we show how to tune the used cost function in coset coding for making trade-off between write energy, endurance, and wordline write disturbance errors. Finally, we assess efficiency of the proposed approach versus existing approaches.

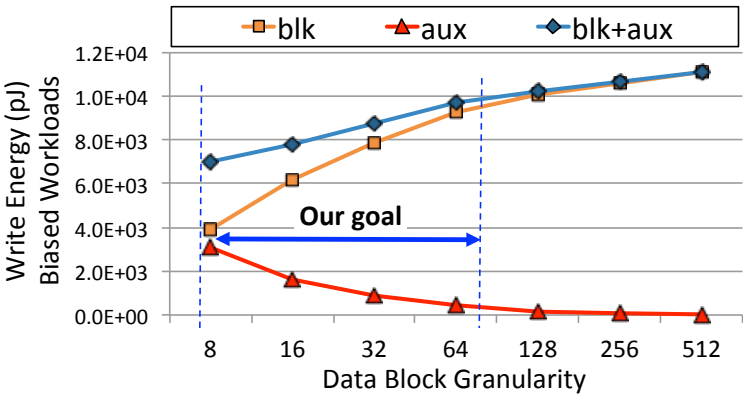
5.2.1 Motivation

Using simulation (see Section 5.2.5), we measure the write energy when the coset encoding proposed in [Wang et al.] is used along with differential write. Figure 33(a) shows the results for 200 million 512-bit random data lines when the encoding granularity ranges between 8 and 512 bits. At a given granularity of x-bits, each x-bits data block is separately encoded using one of six coset (codeword) candidates at the cost of adding 3-bits (two auxiliary symbols in MLC PCM) to identify the candidate used. The figure breaks down the write energy into the energy to write the data and the auxiliary symbols. It shows that when the encoding granularity decreases, the write energy and its dominant component, the data symbol energy, decreases. On the other hand, the auxiliary symbol energy gradually increases and reaches its maximum at the granularity of 8-bits.

We also perform a similar study on real workloads to investigate the relationship between the components of the write energy. Figure 33(b) shows that the energy for the biased workloads is smaller than the random workload case, which is due to data locality. However



(a) Random workloads.



(b) Biased workloads (SPEC2006 and PARSEC benchmarks).

Figure 33: Write energy analysis.

the trend with varying granularity is the same for both workloads. The main reason for energy reduction at small data block granularity is the flexibility of encoding smaller data blocks independently. Unfortunately, this benefit comes at the expense of a high space overhead needed for storing the auxiliary symbols. This overhead reaches 25% at 8-bit granularity (two auxiliary symbols per four data symbols).

The goal of this section is to take advantage of fine-grain encoding granularity while reducing the overhead of auxiliary symbols, using a light weight compression that provides enough space in the memory line to store auxiliary symbols, and making trade-off among the write energy, endurance, write disturbance and area overhead.

5.2.2 Revisiting Coset Candidates

A 4-level cell can be programmed to any one of four resistance states. We denote these states by S1, S2, S3 and S4 and we assume that the states are numbered in the order implied by the energy needed to bring a cell to that particular state, with S1 requiring the least energy and S4 requiring the most energy (see Table 11). Specifically, programming into S1 is done using a RESET pulse, while programming it into S2 is done using a SET pulse, which consumes more energy. Programming into S3 and S4 is done through iterative partial SET pulses [Joshi et al.]. Note that to reach S2, S3, S4, the cell must be first reset before applying the SET pulses. Every two consecutive bits in a memory line are stored in one cell. Hence, an encoding is a particular mapping of the four symbols, '00', '01', '10' and '11' into the four cell states. We assume that the default mapping of the four symbols '00', '10', '11', and '01' is to the states S1, S2, S3 and S4, respectively [Jiang et al.].

The coset candidates used in [Wang et al.] to encode a memory line are based on mapping the two most frequent symbols in a memory line into the two low energy states while maintaining the original data block as much as possible. Assuming that any two of the four symbols can appear more frequently in any particular memory line, the encoding provides $C_4^2 = 6$ different mappings of symbols to states, which is equivalent to using six possible coset candidates in the encoding. Of course, 3-bits (two symbols) are needed for each memory line to record the particular candidate used in the encoding.

Note that the above logic used to select the six coset candidates is suitable for random data since it assumes that in any memory line, any two of the four symbols can appear more frequently in that line. However, it is well documented [Alameldeen and Wood, 2004; Balakrishnan and Sohi; Ekman and Stenstrom, 2005] that in real workloads, the two symbols '00' and '11' appear much more frequently than the other two symbols because many data words contain long runs of 0's or 1's. For example, zero is most commonly used to initialize data, to represent NULL pointers or false Boolean values, and to represent sparse matrices. On the other hand, long sequences of 1's appear in the representation of negative signed numbers. We will take advantage of this knowledge to propose four carefully selected coset candidates and compare the performance of this encoding, called '4cosets,' with that of the

encoding proposed in [Wang et al.], called '6cosets.' Note that by reducing the number of coset candidates from six to four, we reduce the auxiliary information needed to keep track of the coset candidate used from four bits (two symbols) to two bits (one symbol).

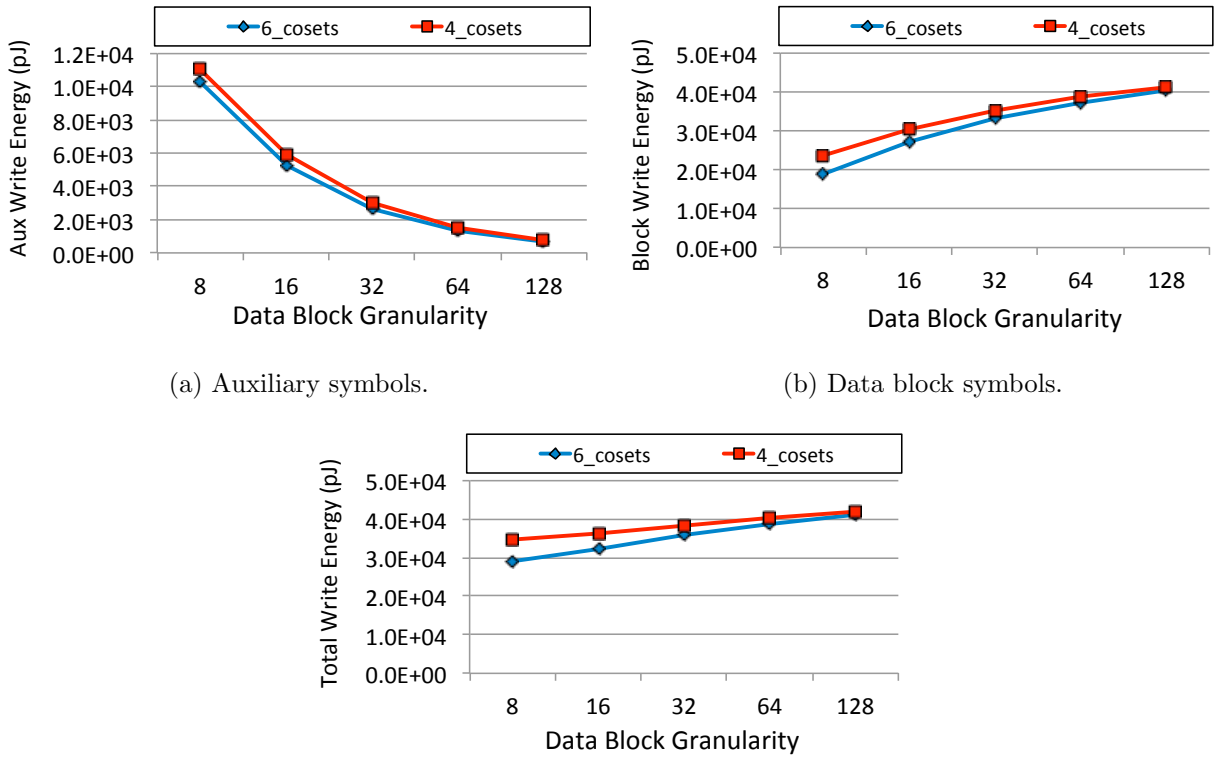
Table 11 shows the symbol-to-state mapping for the four proposed coset candidates. The first candidate, C1, represents the default symbol-to-state mapping. Candidates C2 and C4 map '11' and '00' to the two states with the lowest write energy to take advantage of the fact that sequences of consecutive 0's and consecutive 1's are common in memory traces of real applications. Candidate C3 is chosen so that, when combined with C1, any of the four symbols will be mapped to the two states with the low write energy, either in C1 or in C3. This will be useful for random patterns that do not exhibit any bias.

To compare the effectiveness of the proposed 4cosets encoding with the 6cosets encoding proposed in [Wang et al.], we plot in Figure 34 the write energy for both encodings for 200 million random data blocks with granularity varying from 128-bits down to 8-bits. Because it uses more candidates and has more options for reducing the write energy, 6cosets achieves write energy reduction in the data symbols more than 4cosets. The energy consumption for the auxiliary symbols is also lower for 6cosets than 4cosets despite the fact that 4cosets uses only one auxiliary symbol per data block while 6cosets uses two. The reason is that for 6cosets, we use the six state combinations of the two auxiliary symbols that require the least write energy among the 16 possible state combinations of the two symbols. For 4cosets, all four states of the auxiliary symbol, including the two high write energy states, have to be used to identify the candidate used in the encoding.

The advantage of 6cosets vanishes when we compare the two schemes for real benchmarks

Table 11: Four coset candidates for mapping two bit patterns to the four energy states of a MLC PCM.

State	Write energy [Bedeschi et al., 2009]	Coset candidate mapings of symbols to states			
		Coset C1	Coset C2	Coset C3	Coset C4
S1	36+0 pJ	00	11	11	11
S2	36+20 pJ	10	00	01	00
S3	36+307 pJ	11	10	00	01
S4	36+547 pJ	01	01	10	10



(a) Auxiliary symbols.

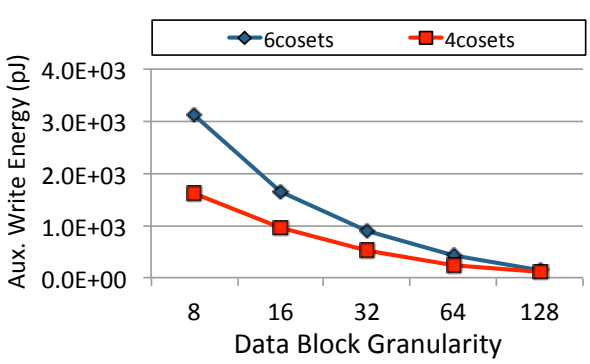
(b) Data block symbols.

(c) Auxiliary + data block symbols.

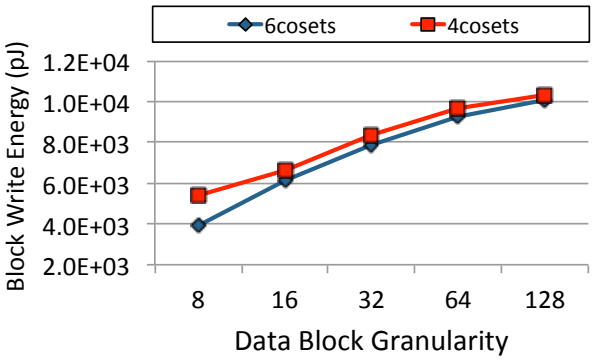
Figure 34: Write energy analysis. The reported write energy is the average for 200 million random data blocks. The PCM memory line is 512-bits.

as shown in Figure 35. The figure shows that 6cosets still has an advantage with respect to the write energy of data symbols. However, the energy to write the auxiliary symbols is lower in 4cosets than in 6cosets because it uses only one auxiliary symbol rather than two, and it uses the two low energy states of the auxiliary symbol to represent the most commonly used coset candidates, C1 and C2. As a result, the total write energy in Figure 35 shows that the two sources of the write energy make a suitable trade-off such that the write energy of 4cosets is almost equal to that of 6cosets for a wide range of data block granularities.

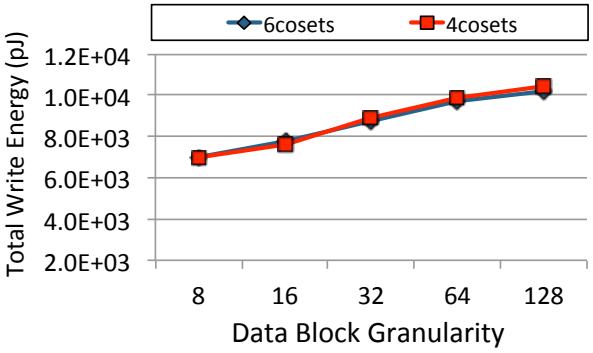
We conclude that both 4cosets and 6cosets consume roughly the same write energy for real workloads. More importantly, 4cosets reduces the number of auxiliary symbols by 50%, which is a large advantage when the memory line is to be compressed to make room for the auxiliary symbols.



(a) Auxiliary symbols.



(b) Data block symbols.



(c) Auxiliary + data block symbols.

Figure 35: Write energy analysis. The reported write energy is the average for SPEC2006 and PARSEC benchmarks. The PCM memory line is 512-bits.

5.2.3 Restricted Coset Coding

In traditional coset encoding, a coset candidate is selected independently for each data block to minimize the write energy for that block. In this section, we introduce a restricted coset encoding which mandates a correlation between the use of coset candidates in a number of consecutive data blocks. This restriction reduces the auxiliary information and does not largely affect the energy minimization capability because the bit patterns of consecutive words are usually similar.

We illustrate the concept of restricted coset encoding by a simple example. Assume that we only use the first three coset candidates, C1, C2 and C3, discussed in Section 5.2.2. Instead of allowing the flexibility of using C1, C2 or C3 independently in each data block,

we can group the cosets into two groups, 'C1,C2' and 'C1,C3', and force any data block in a memory line to either use one of C1 and C2 in the encoding or to use one of C1 and C3. For example for a 16-bit encoding granularity, there are 32 data blocks in a 512-bit memory line. Restricted coset encoding proceeds as follows: 1) use the two candidates C1 and C2 to encode each of the 32 data blocks, 2) use the two candidates C1 and C3 to encode each of the 32 data blocks and 3) select the better of the encodings produced in steps 1 and 2.

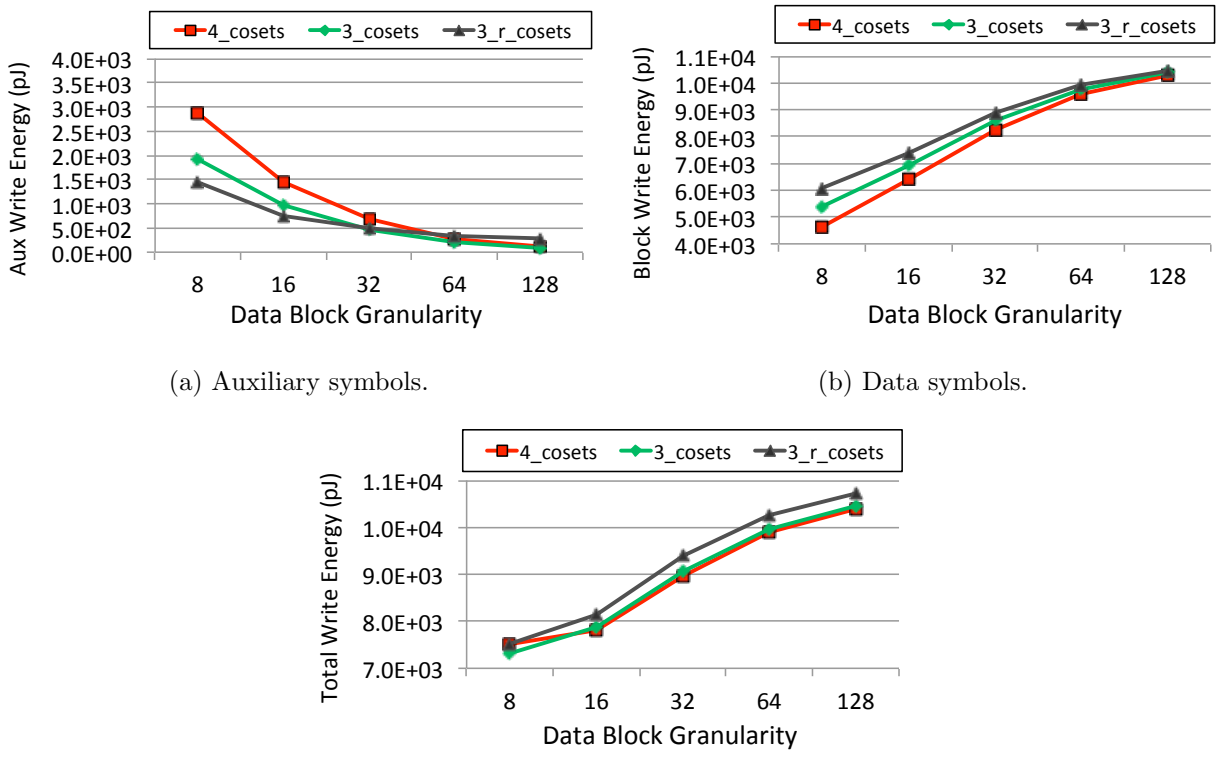
Of course, this restricted method needs one global auxiliary bit per memory line to determine the coset group used in that line, in addition to one auxiliary bit per data block, for a total of 33 auxiliary bits (17 symbols) per memory line. This is fewer than the unrestricted encoding which needs 64 auxiliary bits (32 symbols) per memory line, two bits per data block.

To explain the ramification of restricting the use of cosets, we recall from the previous section the justifications for choosing the three candidates C1, C2 and C3. C2 is useful for biased data with many sequences of consecutive 0's or 1's, while C3 is useful for non-biased data. Because of data locality, we can expect consecutive words in the memory line to either be all biased or not. In the former case, not using C3 will not hurt much, and in the latter case, not using C2 will not hurt much.

To evaluate the effect of restricting the use of cosets on the write energy, we plot in Figure 36 the write energy of 4cosets, 3cosets (that unrestrictedly uses candidates C1, C2 and C3) and the restricted coset coding (called 3-r-cosets). We draw two observations from this figure. First, 3cosets only slightly increases the write energy over 4cosets. Second, reducing the number of auxiliary information by the proposed restricted method increases very little the write energy relative to 4cosets. The main advantage of restricting the coset candidates will be clear in the next section where we use WLC_6 to make room in the memory line for embedding the auxiliary information.

5.2.4 WLCRC: Integrating WLC with Restricted Coset Encoding

In this section, we will use WLC_6, abbreviated to WLC, to make enough room in the memory line to store the auxiliary symbols of the restricted coset encoding. Because of the reduction



(a) Auxiliary symbols.

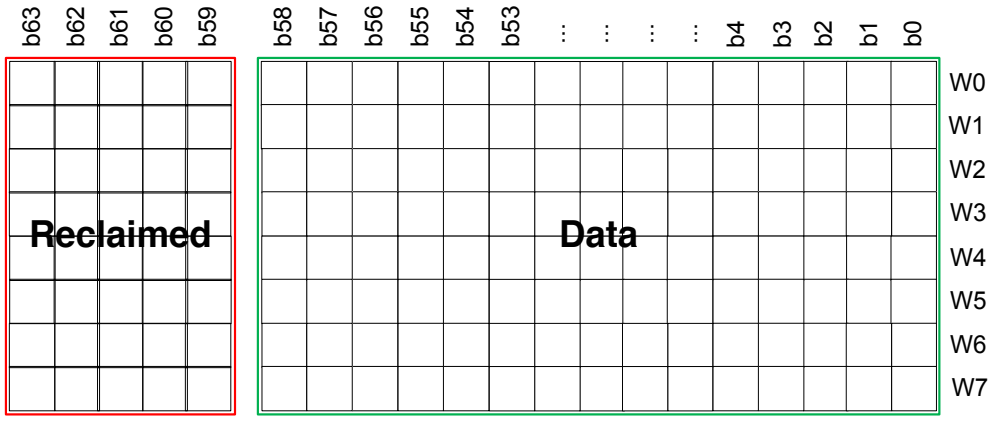
(b) Data symbols.

(c) Auxiliary + data symbols.

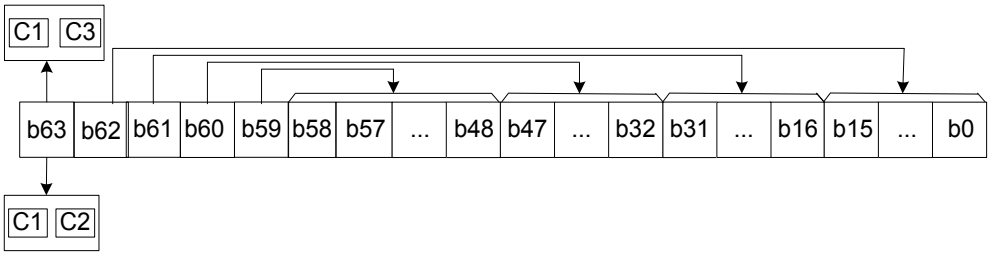
Figure 36: Write energy analysis. The average write energy are reported for the SPEC2006 and PARSEC benchmarks.

in the auxiliary information needed for 3-r-cosets, WLC will be able to provide the necessary room in 90% of memory lines to embed the auxiliary symbols. A global bit (symbol) per memory line will be used to flag the lines that cannot be compressed. Those lines will be written in memory without encoding.

Figure 37(a) shows the WLC that compresses the 6 most significant bits of each 64-bit word of a 512-bit memory line. In this figure, each row represents the 64-bits, '$b^i_{63}, ..., b^i_0$', of each of the eight words, w^i, $i = 0, ..., 7$. WLC compresses the memory line as long as all six MSBs, '$b_{63}, ..., b_{58}$', of **each word** are '000000' or '111111'. Thus, it splits each word into two parts: the five reclaimed bits, '$b_{63}, ..., b_{59}$', and the data bits '$b_{58}, ..., b_0$', which are not changed by the compression. When decompressing the word, bit b_{58} is extended to the reclaimed bits, similar to sign extension. The five reclaimed bits will be used to store the auxiliary bits of the 3-r-coset encoding at 16-bit granularity.



(a) Word Level Compression (WLC).



(b) Restricted coset coding at 16-bit granularity.

Figure 37: Integrating WLC with restricted coset coding.

Figure 37(b) shows the format of the restricted coset encoding at a 16-bit granularity. Specifically, each 64-bit word is divided into 4 data blocks, '$b_{58}, .., .b_{48}$', '$b_{47}, ..., b_{32}$', '$b_{31}, ..., b_{16}$', and '$b_{15}, ..., b_0$'. To record the coset restriction used for encoding each 16-bit data block in a word, we use bit b_{63} to determine which group of $coset_{C1,C2}$, or $coset_{C1,C3}$, is used to encode the four 16-bits data blocks in the 64-bit word. Then, the four bits, '$b_{62}, ..., b_{59}$', are used to identify which coset candidate (restricted by the specified group) is used in each data block.

Algorithm 2 is a pseudo-code for WLCRC-16 where the eight 64-bit words, w^i, $i = 0, ..., 7$, are independently encoded in parallel when the memory line is compressible (Line 1-2). To encode a word, w^i, it is divided into 4 sub-words w^i_j, $j = 0, ..., 3$ (Line 3), and the sub-words are encoded in parallel using the three cosets C1, C2 and C3. Then, the energy cost, $cost_k(w^i_j)$, of encoding w^i_j using Ck is computed for $j = 0, ..., 3$ and $k = 1, 2, 3$. This allows the estimation of $cost_{1,2}(w^i)$ and $cost_{1,3}(w^i)$, which are the costs of encoding w^i using C1 or

Algorithm 2: Pseudo-code for WLCRC-16 applied to a compressible memory line.

1 **begin**
2 **for** $w^i, i = 0, ..., 7$, *in Parallel,* **do**
3 Divide w^i into four sub-words w^i_j, $j = 0, ..., 3$
4 Encode w^i_j using C1,C2 and C3 in parallel $cost_{1,2}(w^i) = \sum_{j=0}^{3} min\{cost_1(w^i_j), cost_2(w^i_j)\}$ $cost_{1,3}(w^i) = \sum_{j=0}^{3} min\{cost_1(w^i_j), cost_3(w^i_j)\}$
5 If $(cost_{1,2}(w^i) < cost_{1,3}(w^i))$ encode w^i using C1/C2 else encode w^i using C1/C3.

C2 and using C1 or C3, respectively (line 4). Finally, the encoding with minimum cost is selected to be written in memory.

Note that, driven by the compressed format, we applied the coset restriction to the data blocks in a 64-bit word, rather than to the entire memory line, as described in the previous section. Hence, our proposed encoding, called WLCRC, applies only to data blocks at granularities of 8, 16, 32 and 64 bits. However, to apply WLCRC at 8-bit granularity, eight bits must be reclaimed by WLC from each word. To apply it at 32-bit granularity, only three bits must be reclaimed. At 64-bit granularity, WLCRC is identical to the unrestricted 3cosets encoding in which also two bits need to be reclaimed.

Finally, we note that WLC can be integrated with unrestricted 3cosets or 4cosets encodings, as long as WLC can reclaim enough bits to embed the auxiliary bits for the encoding. For example, to use WLC with 4cosets at data block granularities of 8, 16, 32 or 64 bits, WLC has to reclaim 16, 8, 4 and 2 bits per word, respectively. Note, however, that according to Figure 32, as long as the number of reclaimed bits per word is less than or equal to 6 , WLC compresses 90% of the memory lines. Otherwise it compresses fewer than 55% of the lines. In summary, *the selection of data block granularity and restricted/unrestricted encoding is a trade-off between the encoding overhead and the write energy reduction.*

5.2.4.1 WLCRC Architecture Figure 38 shows the on-chip architecture of WLCRC compression+encoding and decoding+decompression for a data block granularity of 16. The 512-bit line from the memory controller is sent to the WLC module to check whether it is compressible or not. If the line is compressible, WLC enables the encoder to activate eight restricted coset encoding modules. When differential write is used, each compressed 64-bit word out of WLC is differentiated with the corresponding 64-bits from the currently stored memory line and the difference is used in a restricted coset module to compute the encoded word to be written into memory. If WLC cannot compress the data line, the uncompressed, unencoded line is compared with the memory current data and the difference is written to memory. One auxiliary bit is used to differentiate encoded from non-encoded lines, which means that an additional symbol must be stored with the 256 symbols of the memory line. Consequently, the total encoding space overhead is $< 0.4\%$. Note that the leading 6cosets scheme stores two auxiliary symbols with each memory line, which is double the space overhead of WLCRC.

The eight 64-bit encoders operate in parallel. Each encoder splits the word into 16-bit blocks and for each block, the writing cost is estimated when each of the candidates, C1, C2 or C3 is used for mapping the symbols to the cell's states. Note that to encode the four data blocks in parallel, the most significant block, '$b_{58}, ..., b_{48}$', contains 11 bits rather than 16 bits since bits '$b_{63}, ..., b_{59}$' are not known before the encoding. It is possible, however, to consider all 16 bits, '$b_{63}, ..., b_{48}$', in the encoding process if we encode the most significant block after the encoding of the other three blocks is completed, which will increase the encoding (and similarly the decoding) delay. We chose the fully parallel solution.

The decoding follows the reverse structure of the encoding. Specifically, it first checks whether the memory line has been compressed/encoded or not. If yes, the decoder decodes the eight words and then a WLD module decompresses the decoded words. The decoding process is simple as the most significant bit of each 64-bit word, b_{63}, determines the coset group that had optimized that word in the encoding process. Then, the four bits, '$b_{62}, ..., b'_{59}$ determine the coset candidate that should be used to decode the corresponding 16-bit block.

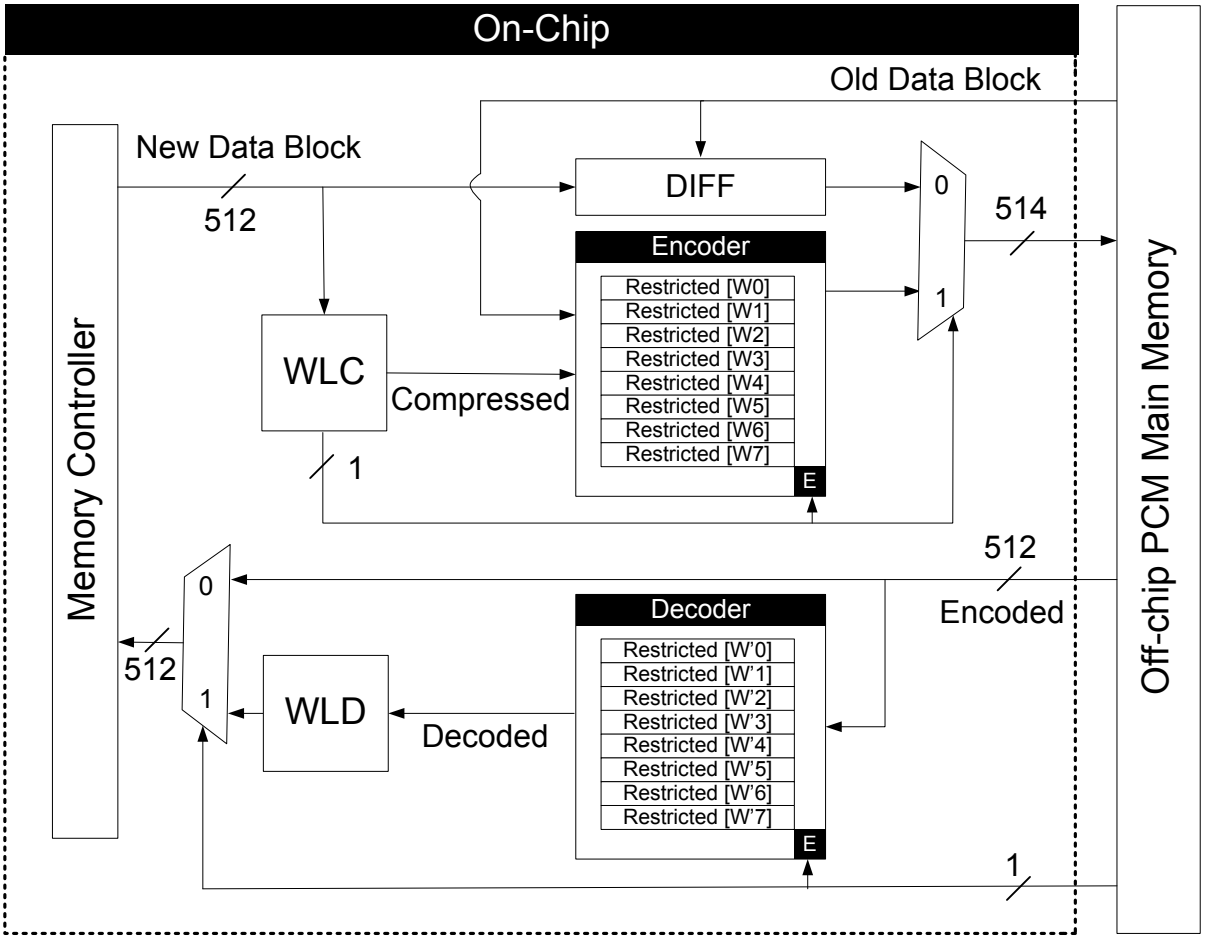


Figure 38: On-chip WLCRC architecture for 16-bit granularity.

5.2.4.2 Hardware Overhead In this section, we evaluate the delay, power, area, and energy of WLCRC-16. Verilog implementations were synthesized using Synopsys Design Compiler targeting a 45nm FreePDK standard cell library [FreePDK45]. The WLCRC implementation assumes 512-bit memory lines requiring eight encoding modules to simultaneously encode the compressed words by WLC. We assume that the additional encoding bits added to the 512-bit memory line are handled through a wider main memory interface. Our results show that the WLCRC modules incur an area overhead of $0.0498mm^2$, which is negligible compared to the PCM main memory area. The delay of WLCRC modules is $2.63ns$ and $0.89ns$ during a write and read, respectively. The energy consumption of the WLCRC modules is $0.94pJ$ and $0.27pJ$, per write and read memory line access, which is negligible

compared to the write energy consumed by cell programming. Note that the WLC compression/decompression portion of the design is very small compared to the encoding/decoding unit, requiring only $0.0002mm^2$ area, $0.13ns$ delay, and $0.0017pJ$ of energy.

5.2.5 Experimental Settings

To conduct experiments, we developed a trace driven simulator. The input traces to our simulator were collected with Virtutech Simics [Magnusson et al., 2002]. As it is widely assumed that PCM employs differential write, or writing bits only when the value differs from the previously stored value, for each memory write transaction the traces store both the value to be stored as well as the value to be overwritten.

For trace generation, our simulations assume an 8-core 4GHz chip multiprocessor. Each core has a 2MB private L2 cache. We model a 32GB PCM main memory with two channels; each channel has two DIMMs and each DIMM has eight chips and 16 banks. In general, the read queue is given a higher priority than the write queue. However, to avoid starvation, when the write queue exceeds 80% of capacity, writes are serviced ahead of reads. For write energy evaluation, we scaled the write energy reported from an MLC PCM prototype at the 90nm process node [Bedeschi et al., 2009; Wang et al.]. All studied schemes are implemented on top of differential write [Zhou et al., 2009]. We used a 'single' RESET and multiple SET iteration-based programming strategy [Braga et al., 2010] to increase programming accuracy in our evaluation[2]. If the cell value does change and requires a write, it consumes the RESET energy of about $36pJ$. Then depending on the cell value, SET operations may ensue to change its resistance requiring between $20pJ$ and $547pJ$.

The write disturbance error rates (DER) of MLC PCM states when the adjacent cell is being written (modeled by the RESET operation) are also extracted from the literature [Jiang et al.]. Thus, an idle cell in the minimum resistance state is assumed to be error free as the high heat of the RESET process will not increase its resistance. Note that the lowest energy states, S1 and S2, are the highest and lowest resistance states, respectively. RESET places

[2]An alternative programming scheme is to use one SET pulse and multiple RESET pulses [Joshi et al.]. Because of reliability concerns such as resistance drift and difficulty in controlling the melting process, we selected the 'one SET - multiple RESETs' scheme.

Table 12: System configuration

CPU	8-core, 4GHz, single-issue	
L2 Cache	private 2MB, 8-way 64B line, write-back	
32GB PCM Main Memory	2 channels 2 DIMMs per channel 16 banks per DIMM, 32-entry, 64B line write pausing scheduling	
MLC PCM 36pJ RESET Energy	Set Energy [Wang et al.]	Disturbance Rate [Jiang et al.]
	S1: 0pJ	DER: 12.3%
	S2: 20pJ	DER: 0.0%
	S3: 307pJ	DER: 27.6%
	S4: 547pJ	DER: 15.2%

the cell in the highest resistance state (S1) and a short high write current can place the cell in the lowest resistance state (S2) (immune to write disturbance), similar to SLC PCM. States S3 and S4 require many more precise SET operations to achieve a resistance between the high and low energy state, making them require high write energy as well as making them susceptible to write disturbance when idle. All schemes are compatible with the standard 'Verify-n-Restore' approach [Dong and Xie] to correct disturbance errors that may have occurred. Detailed simulation parameters are recorded in Table 12.

To evaluate endurance, we counts the average number of updated cells per write request since fewer RESET operations leads to higher cell endurance. To evaluate write disturbance, we count the number of idle cells disturbed by neighboring cells that need to be updated in the write request. The write disturbance happens during the RESET process that generates high heat and can potentially disturb adjacent cells in states S1, S3 and S4 with the probabilities shown in Table 12 based on a 22nm technology node [Jiang et al.].

5.2.6 Workloads

In order to study the impact of our scheme on write energy of MLC PCM, we selected memory intensive workloads. In particular, we include twelve write-intensive benchmarks from SPEC CPU2006 and supplement them with *canneal* from PARSEC. We selected only the canneal workload from PARSEC because most PARSEC benchmarks are computation intensive and in most cases also have a very small memory footprint. To be consistent with the SPEC CPU workloads, canneal was executed in our experiments in single-threaded mode and with the largest 'native' data input that resulted in a 940MB memory footprint. For SPEC CPU2006, we use the large 'reference' inputs that are designed to stress the system.

5.2.7 Evaluation

To evaluate the effectiveness of WLC and the restricted coset coding, we compared the following schemes:

Baseline: This scheme just uses standard differential write for energy reduction when writing a 512-bit memory line into MLC PCM.

FlipMin [Jacobvitz et al.]**:** This scheme uses two symbols per memory line for 16 coset candidates, generated using the technique in [Seyedzadeh et al., 2016b], operating on a 512-bit memory line. Note that this scheme, as well as the next scheme, FNW, were proposed for SLC PCM and were adapted in our implementation for MLC PCM.

FNW [Cho and Lee]**:** This scheme selects the original data block or its complement, depending on which one uses less write energy. A single auxiliary bit is enough to indicate that a data block is complemented. Thus, to match the space overhead of FlipMin which uses two symbols (four auxiliary bits) per 512-bits memory line, we partition the memory line into 128-bit blocks that can be inverted independently with FNW.

DIN [Jiang et al.]**:** This scheme uses a 3-to-4-bit code word mapping to remove high energy states. Write disturbance errors are mitigated by a 20-bit BCH code to correct two write errors in the write verification process. To avoid the space overhead of this encoding, it

is applied only to 512-bit memory lines that can be compressed with FPC+BDI to at most 369 bits. DIN was originally proposed to reduce write disturbance.

6cosets [Wang et al.]**:** This scheme uses six coset candidates to map any two of the four symbols to the low energy states S1 and S2. Thus, it also incurs a space overhead of two auxiliary symbols (four bits) per 512-bit memory line.

COC [Kim et al., b] +4cosets: This scheme uses COC along with directly applying the four coset candidates shown in Table 11. The encoding is applied at 16-bit or 32-bit granularity for lines that are compressed to at most 448 bits or 480 bits, respectively.

WLC+4cosets: This scheme uses WLC along with directly applying the four coset candidates shown in Table 11. It requires a space overhead of one symbol per memory line to indicate if the memory line is compressible or not. Unless stated otherwise (in Section 5.2.8), the default WLC+4cosets encoding granularity is 32-bit blocks.

WLCRC: This scheme, WLC with the restricted coset encoding, uses the first three coset candidates shown in Table 11. The default WLCRC granularity is for 16-bit blocks, denoted as WLCRC-16.

Note that for COC+4cosets, WLC+4cosets and WLCRC-16 encoding techniques, when COC and WLC cannot sufficiently compress the block, the original, uncompressed 512-bit memory line is written. Because the auxiliary symbol must only record the compression state, even though it can store four states, we select only the two lowest energy states for this purpose. Moreover, since COC and WLC compress more than 90% of memory lines, we flagged the 'compressed' state with the lowest energy state.

In the following sections, we compare these enumerated schemes for their write energy, their average number of updated cells per write request, and their average number of write disturbance errors per write request for as close to an ISO-overhead comparison as possible. In general, these schemes are categorized into two groups. The first group, including FlipMin, FNW, and 6cosets, augments the encoding space for an entire memory line to reduce the energy, while the second group, including DIN, COC+4cosets, WLC+4cosets, and WLCRC-16, use compression techniques in order to allow encoding at a finer block granularity to reduce write energy.

5.2.7.1 Write Energy Figure 39 compares the write energy for different schemes. While FNW is superior to FlipMin, in part due to its ability to operate on a smaller block size, 6cosets performs the best among the schemes designed to operate on the full memory line. Interestingly, on average DIN, which operates on the smallest block size, performs close to 6cosets, but its effectiveness is much more benchmark dependent. This is likely due to the varied effectiveness of the FPC+BDI compression that enables DIN encoding within each particular workload. In contrast, word level compression is extremely effective and consistent in reducing energy. In particular, WLC+4cosets provides a 46% improvement over the baseline and 32% improvement over the leading 6cosets approach. Further decreasing the block granularity at the expense of the coset flexibility provides a significant additional improvement. WLCRC-16 reduces the write energy by 10% over WLC+4cosets and increases the improvement over the baseline to 52% while providing an overall improvement of 39%, 39%, and 48% versus 6cosets, DIN, and FlipMin, respectively. For all workloads, including non-intensive memory applications, WLCRC-16 reduces the write energy on average versus other schemes. Moreover, Figure 39 shows that, as expected, write energy grows considerably for intensive workloads, such as *milc*, *lesl*, and *sopl,* while the effectiveness of WLC and, in particular, WLCRC-16 scales very well. For the high energy benchmark *wrf* where 6cosets is not effective but DIN is effective, WLC-based schemes are still the best approach.

The effectiveness of the proposed techniques comes from several factors. First, they employ coset candidates that best map commonly occurring bit sequences to low energy

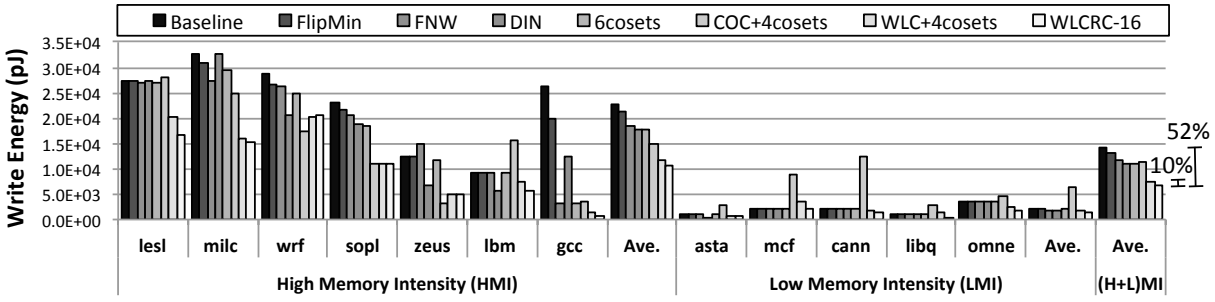


Figure 39: Comparison of write energy for various schemes on SPEC CPU2006 and PARSEC inputs.

states for different types of workloads. Second, WLC+4cosets and WLCRC achieve a small data block granularity for encoding, which can more precisely select the best coset candidates to map symbol encoding. Third, WLC compression can be applied to more than 90% of the memory lines in these representative workloads, making coset encoding possible in a very high percentage of blocks. Fourth, contrary to compression techniques that significantly change the content of compressed data blocks even for relatively small changes in actual data, WLC only compresses a small fraction of the 64-bit word to create room for the coset auxiliary bits, retaining much of the temporal locality that makes differential writes effective. In contrast 6cosets and FlipMin operate at a large data block granularity (512-bits) since they require a substantial increase in auxiliary information to operate at a granularity similar to WLC-based encoding. The additional auxiliary bits tend to work against the energy saved in the data block due to the random nature of the encoding. For example, decreasing the granularity for 6cosets from 512-bits to 16-bits increases the write energy ratio of the auxiliary symbols to the data symbols from 0.78% to 12.5%. The restricted coset method further decreases the number of auxiliary symbols, making encoding improvements to the data block more impactful.

In contrast to DIN, which requires 25% compression of the memory line to accommodate 3-bit to 4-bit expansion, restricted coset encoding requires only 7.8% compression. Figure 32 shows that more than 70% of memory lines cannot be compressed for DIN while 90% of memory lines are compressible with WLC. Moreover, the compression and BCH encoding employed by DIN increase symbol flips in the memory line, limiting the possible energy savings.

The 10% write energy reduction of WLCRC-16 versus WLC+4cosets is primarily due to the latter's need to operate on 32-bit blocks. For WLC+4cosets to operate at 16-bit granularity would require WLC to reclaim eight bits every 64-bit words rather than five bits for WLCRC-16. Unfortunately, the number of compressible memory lines reduces from 90% to 48% when eight rather than five bits are to be reclaimed by WLC, making WLCRC-16 much more effective.

While COC+4cosets is somewhat effective in reducing the write energy for high memory intensity workloads, it tends to increase the write energy for low memory intensity workloads.

Our analysis of the COC-4cosets encoded memory lines shows that it uses 16-bit data block granularity for most write requests. However, since PCM uses differential write to update only modified bits, it is important to ensure that compressors not increase the bit entropy of consecutive write requests. Because COC was not designed to preserve bit entropy, it often switches between the 28 different compressors, changing the data bit patterns from the original. In contrast, WLCRC-16 does not change the bits in the data except in only a few locations, which allows the differential write to take advantage of data locality. This is why, on average, WLCRC-16 uses 39% less energy than COC+4cosets.

In summary, novelty of the WLC is that it is a simple compression mechanism that can, with high probability, compress memory lines enough to make room for auxiliary encoding bits, while preserving the bit location/locality of most of the bits. This is a crucial property for effective differential writes.

5.2.7.2 Endurance PCM main memory employs differential write to decrease the number of written cells primarily to save energy. However, the reduced numbers of writes also benefits endurance. Reducing the number of cells that are changed through intelligent encoding, such as WLC+4cosets and WLCRC-16, can further improve endurance. Figure 40 shows the average number of updated cells per write request. It shows that WLCRC-16 reduces the number of updated cells by 20%, 17%, 16% and 11% versus the baseline, FlipMin, COC+4cosets and 6cosets schemes, respectively. However, the improvement or degradation in endurance varies highly for different benchmarks. For some workloads, such as *wrf*,

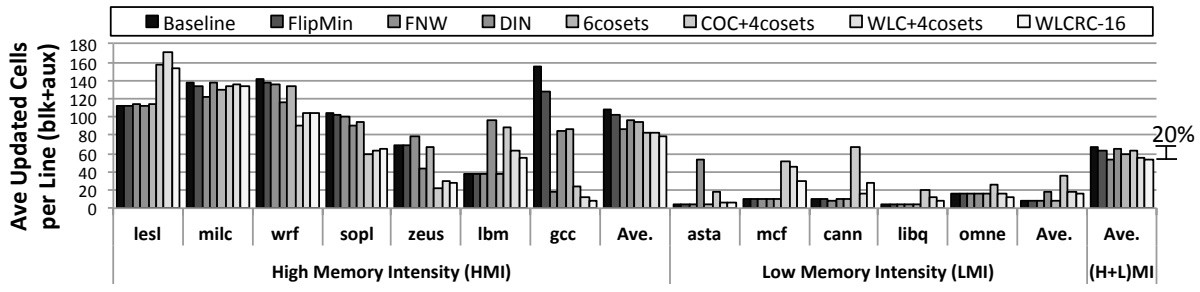


Figure 40: Average number of updated cells per memory line for SPEC CPU2006 and PARSEC inputs.

zeus, *gcc*, and *sopl*, WLCRC-16 not only reduces write energy but also reduces the average number of updated cells, thus improving endurance. For other workloads such as *lesl*, *lbm*, *mcf*, and *cann*, WLCRC-16 more frequently maps high energy states to low energy states to reduce write energy but causes an increase in the number of updated cells compared to other schemes, thus harming endurance. Therefore, WLCRC-16 makes a trade-off between write energy and the number of updated cells for this group of workloads. However, on average WLCRC-16's endurance is considerably better than 6cosets, COC+4cosets and DIN and is on par with FNW.

5.2.7.3 Write Disturbance Write disturbance errors occur during the RESET process. The high heat of RESET (melting the material) can change the resistance of nearby idle cells that are not part of the actual write request. Write disturbance is unidirectional, so it can only decrease the resistance of other cells. Cells with the minimum resistance (S2) are thus immune to write disturbance. However, any RESET operation adjacent to a cell in states S1, S3, or S4 may still incur write disturbance.

Our results shown in Figure 41 indicate that all schemes on average face write disturbance errors ranging from three to four every request to write a 512-bit memory line. For more memory intensive workloads such as *lesl* and *milc*, the average number of write disturbance errors across all schemes ranges between seven and nine. DIN compressed data blocks increase the number of cells written which increases write disturbance to be the highest among all the approaches. However, its 20-bit BCH code offsets this somewhat by correcting two disturbance errors. WLC+4cosets and WLCRC perform generally well, averaging around the minimum point for all benchmarks.

Part of the trends observed in Figure 41 is the correlation between disturbance faults and the number of updated cells per write operation. When more cells are written, the likelihood of disturbing adjacent idle cells increases.

Since PCM uses differential writes, a memory line is always read before it is written. This allows for the detection of write errors by a "read-after-write" process, thus avoiding silent data corruption (SDC) due to write disturbance. It also allows for an iterative verify-and-restore (VnR) process [Dong and Xie] which iterates until data is correctly written, thus

eliminating detected uncorrectable errors (DUE). Consequently, any available Chipkill capability can be used for non-write-disturbance errors. It was shown in [Jiang et al.] that write disturbance errors can be completely removed if 3-5 iterations of VnR are used. Moreover, only the cells that are neighbors of the written cells are involved in each VnR iteration, which limits the effect on memory bandwidth and avoids resource starvation. As indicated in [Jiang et al.], minimizing the probability of write disturbance (which WLCRC does) will improve performance because of the reduction in the number of VnR iterations. Finally, we note that although the different schemes differ in the average number of disturbances per line, the maximum number of disturbances per line changes very little across schemes.

In summary, WLCRC improves write energy while achieving comparable endurance and write disturbance compared to the schemes specifically designed to improve these metrics.

5.2.7.4 Multi-objective Optimization in MLC PCM The results in Figures 39 and 40 show that, for some applications with unbiased patterns, such as *lesl* and *lbm*, minimizing the write energy may increase the number of updated cells and result in degraded endurance. The main reason is that sometimes the coset candidate that minimizes energy actually increases the number of cells written into low energy states to avoid a relatively smaller number of writes into high energy states. It is possible, however, to select the encoding cosets based on a function that combines energy and endurance, thus sacrificing some energy improvement to attain better endurance. For example, recalling line 5 of Algorithm 2, if

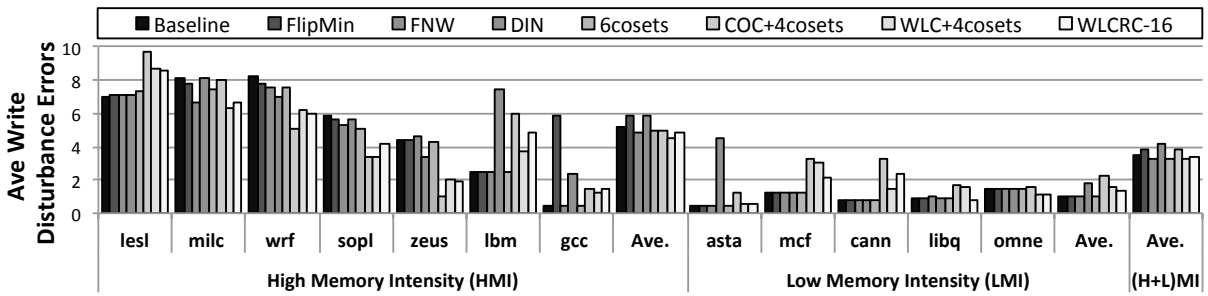


Figure 41: Average number of disturbance errors per memory line for SPEC CPU2006 and PARSEC inputs.

the difference between $cost_{1,2}(w^i)$ and $cost_{1,3}(w^i)$ is smaller than a threshold, T, then the encoding choice can be made based on the number of written symbols rather than energy.

We applied this multi-objective scheme to WLCRC-16 and successfully improved the endurance with a negligible sacrifice in energy saving. For example, when WLCRC-16 with T=1% is applied to *lesl* and *lbm*, the average number of updated cells is reduced from 153 to 133 and from 55 to 49, respectively, while the write energy increased by less than 1%. When we applied WLCRC-16 with T=1% to all benchmarks, the number of updated cells decreased by 19% (52 to 42) on average, while increasing the write energy from 6777pJ to 6885pJ. Relative to the baseline, applying the multi-objective optimization to WLCRC-16 increases the endurance improvement from 20% to 35% while resulting in a nominal degradation of the write energy improvement from 52% to 51.4%, on average.

5.2.8 Sensitivity to Granularity

To better understand the interaction between WLC and coset encoding, we analyze the impacts of data block granularity on write energy, the number of updated cells and write disturbance errors. To clarify the difference of reducing one coset candidate and restricting the coset configurability, we also include a 3cosets approach that is as flexible as 4cosets from an encoding perspective but has the same coset candidates as the restricted coset (C1-C3 in Table 11). We report separately the energy to write the auxiliary and the data symbols.

5.2.8.1 Impact of Granularity on Write Energy Figure 42 shows the write energy when WLC is used with 4cosets, 3cosets and 3-r-cosets for four data block granularities. WLCRC-16 (restricted coset with 16-bit block size), achieves the minimum write energy of 6777pJ on average of all the workloads. This is 10% and 11% lower than 4cosets and 3cosets, respectively, at their minimum energy point, which is for a data block granularity of 32 bits.

To understand why 4cosets and 3cosets require more energy for a 16-bit data block than at a 32-bit block size, as well as more energy than WLCRC at a 16-bit block size, we examine the percentage of compressed memory lines by each scheme. Recall that WLCRC-16 uses one global auxiliary bit per memory line and five auxiliary bits to encode the four independent

blocks of each 64-bit word. Thus, WLCRC-16 requires only six bits of compression per word to be applied and this allows for 90% memory lines to be encoded. In contrast, 4cosets and 3cosets at 16-bit granularity force WLC to provide 8-bits of storage for auxiliary bits in addition to 1 bit for the compressed leading bits, requiring the reclamation of 9-bits. As a result, the percentage of lines that can be encoded drops to 48%. Of course, since both 4cosets-32 and 3-coset-32 require five bits to store auxiliary bits in the reclaimed part, WLC can be applied on 90% of lines. This advantage to the application of compression outweighs the encoding advantage of 4cosets-16 and 3cosets-16, respectively, making 32-bit block granularity the minimum energy point for those approaches.

Of course WLCRC-32 is less effective than WLCRC-16 because it can be applied to the same number of memory lines, but has a coarser granularity of encoding that is less flexible for achieving low-energy states. To further increase the granularity to an 8-bit data block, the reclaimed part must grow to include more than eight bits. Specifically, WLCRC-8 requires seven auxiliary bits to split the word into seven parts noting that the most-significant (eighth) byte will need to be compressed away using WLC to reserve space for the auxiliary bits. When combined with the restricted auxiliary bit, WLC compresses five symbols per word for WLCRC-8, which is only possible for 46% of memory lines. The flexibility of encoding cannot offset the lost compression effectiveness relative to WLCRC-16. -.05cm

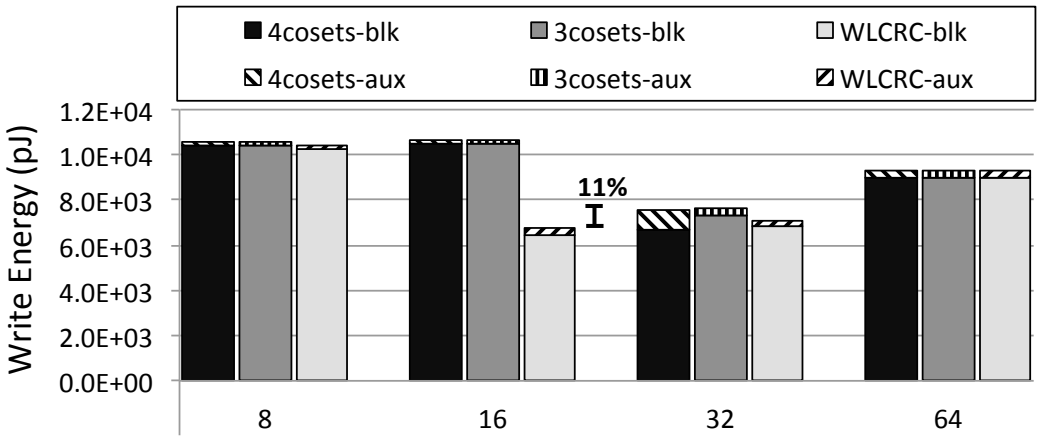


Figure 42: Write energy comparison for four different data block granularities 8, 16, 32, and 64.

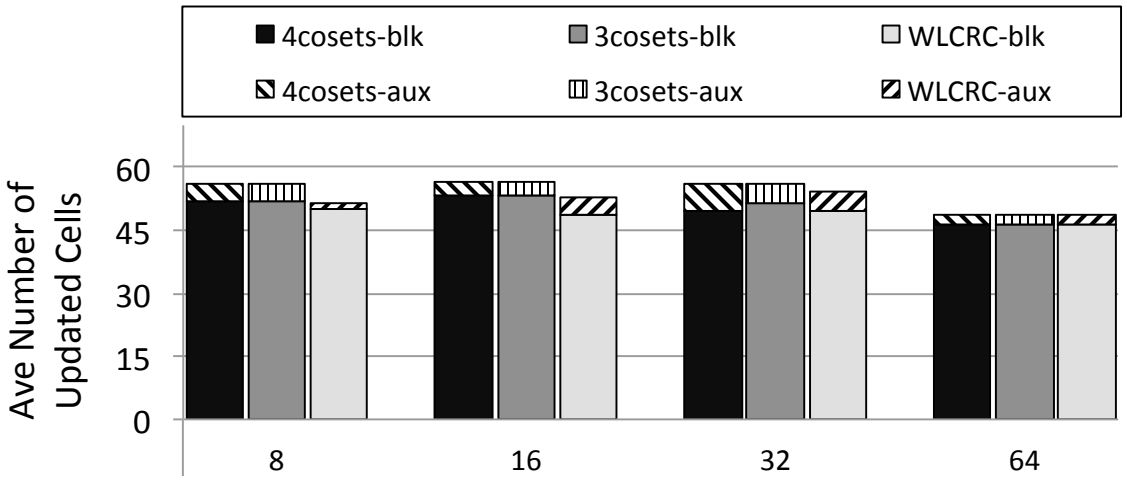


Figure 43: Comparison of average updated cells per memory line for four different data block granularities: 8, 16, 32, and 64.

Figure 42 also breaks down the write energy into energy from the auxiliary part and the data part, independently. Note that the average data block energy is reported based the average write energy of data blocks of compressed words + incompressible memory lines. The auxiliary part reaches a maximum of 5.5% of the total write energy for WLCRC-16, which is less than the 7.8% of the space the auxiliary part requires. The main reason for the low write energy of the auxiliary part is that the restricted cosets incur less bit changes in the auxiliary part compared to unrestricted cosets. When group cosets switch between C1, C2 and C1, C3 (as shown in Table 11), the coset candidate C1 which is the most frequent coset, exists in both groups. We allocate the auxiliary bit '0' to the coset candidate C1 that causes the most symbols in the reclaimed part to remain in the low energy states of S1 or S2. For 4cosets, we allocate energy states S1, S2, S3 and S4 to coset candidates C1, C2, C3 and C4. Since coset candidates C1 and C2 are the two most frequent candidates, it keeps the auxiliary part in the low energy states, S1 and S2, for the most of the write requests. 3cosets does not employ C4 (S4) similarly minimizing the high energy states. *We conclude that the use of WLC to make space for encoding auxiliary bits in the reclaimed part is effective for minimizing write energy. Moreover, the selection of WLCRC-16 is supported as the best trade-off of encoding and block size granularity to minimize write energy.*

5.2.8.2 Impact of Granularity on Endurance Figure 43 shows the number of updated cells (a metric of endurance) as data block granularity scales. At 16-bit granularity, WLCRC reduces the number of updated cells by 8%, on average compared to WLC+4cosets and WLC+3cosets. In this case, for smaller block granularities (i.e., eight bits) the restricted coset reduces the number of updated cells. For example, at 16-bit granularity, the average number of updated cells in a memory line for WLCRC is 10% less than for WLC+4cosets and WLC+3cosets, while the auxiliary parts update roughly the same number of cells. As the data block granularity increases to 64, all schemes require similar number of updated cells, which is about 10% fewer than WLCRC-16.

5.2.8.3 Impact of Granularity on Disturbance Figure 44 shows the average write disturbance errors for different data block granularities. The average write disturbance errors is approximately three per memory line. However, when data block granularity becomes more coarse, the number of symbol flips decreases, which results in fewer write disturbance errors. One observation from this figure is that the data blocks incur a considerably higher number of write disturbance errors compared to the auxiliary part for WLC-based techniques. This is due to the incidence of 25% bit flips, on average, of the data block. However, the disturbance

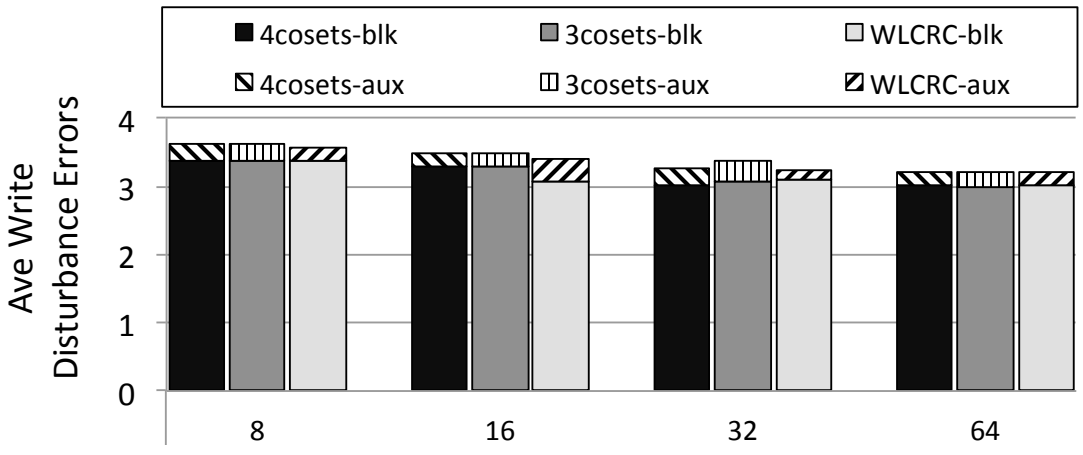


Figure 44: Comparison of the write disturbance errors per memory line for four different data block granularities: 8, 16, 32, and 64.

errors from the auxiliary bits in the reclaimed part do not change dramatically across different block sizes, as the larger reclaimed parts are only applied when WLC is successful.

5.2.9 Sensitivity to Energy Levels

The analysis in this paper is based on the MLC PCM write energy levels previously reported in the literature [Bedeschi et al., 2009; Wang et al.] shown in Table 12. However, subsequent improvements to MLC PCM devices along with better iterative programming approaches may have significantly reduced the energy of writing to intermediate states. To estimate the effect of these write energy improvements on the effectiveness of WLCRC-16, we repeated our experiments with the write energy to high energy states (i.e., S4 and S3) reduced as reported in Figure 45, while keeping the energy of S1 and S2 unchanged. The results show that when the write energy cost of these high energy states is reduced by more than 6×, WLCRC-16 still reduces the write energy by 32% relative to the baseline.

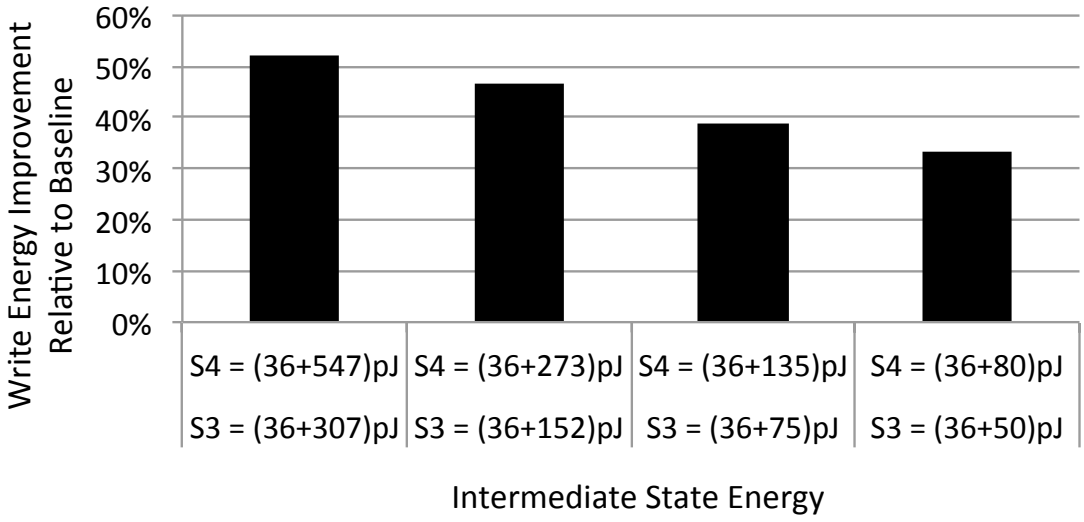


Figure 45: Sensitivity of WLCRC-16 to energy levels.

5.3 REDUCING WRITE DISTURBANCE IN SLC PCM

As the proximity of bitlines and wordlines considerably reduces, both bitline and wordline write disturbances constitute the main sources of unreliability in super dense PCM. Our goal in this section is to tackle these critical reliability challenges. To this end, we use the proposed multi-tiered compression in Section 5.1 and modify coset coding in Section 5.2.3 to minimize the number of aggressor cells that disturb victim cells in the active wordline and the corresponding neighboring wordlines as shown in Figure 4. When the number of aggressor cells reduces in super dense PCM, the number of extra write operations required for eliminating write disturbance errors decreases that leads to performance improvement.

5.3.1 Coset Coding vs. Pointer Approach

Coset encoding uses a translation function to map each data block into multiple codeword candidates. The codeword candidate that minimizes a cost function is then selected and written into the system and auxiliary bits are used to record which translation function was used. To develop a coset approach to minimize write disturbance, we can take advantage of write disturbance asymmetry in PCM, as some cells have a high probability of disturbance and others are "safe." Specifically, the cost function we propose optimizes system efficiency by minimizing the number of aggressor cells in proximity to potential victim cells.

First we partition a 512-bit cacheline into eight 64-bit datawords $D_0, D_1, ..., D_7$ and encode each of the datawords independently. To encode each dataword D_i, we divide it into equal-size sub-dataword $D_{i,j}$ where $0 \leq i \leq 7$ and $0 \leq j \leq 3$. Then, each 16-bit sub-dataword picks out either the original value or its complement depending on which one minimizes likelihood of incidence of write disturbance. Note that the complementary sub-dataword is obtained by XORing each bit with '1.' Furthermore, each sub-dataword requires precisely one auxiliary bit to retrieve the original data sub-partition in the decoding process. Thus, the encoding incurs 1/16=6.25% area overhead.

To solidify the idea of coset coding for write disturbance reduction in the active wordline and the corresponding neighboring wordlines, we show an example using 16-bit data block

shown in Figure 46(a). When the corresponding dirty cacheline in the last level cache is evicted, we describe only these particular 16 bits (Figure 46(b)) to show how it is encoded and written in the main memory. The modified data block has four aggressor cells that can probabilistically disturb neighboring cells because all aggressor cells must be reset to be written, which produces heat that can affect neighboring cells. Moreover, each aggressor cell is adjacent to four potential victim cells due to their stored '0' values. Instead of writing the data block $W_{n(new)}$ (Figure 46(b)), if the complement of the data block $W_{n(new)}$ (Figure 46(c)) is written instead, it only includes one aggressor cell. Furthermore, this aggressor can only potentially disturb two victim cells as the neighboring cells within the wordline are '1.' This example shows how using coset coding can reduce the number of aggressor cells. However, the encoded data block requires one auxiliary bit per 16 bits to indicate whether the data block or its complement is used to be written in the physical cells. For the typical 512-bit cacheline, this encoding requires 32 auxiliary bits.

While coset encoding can eliminate the potential for many disturbance errors, like in

W_{n-1}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$W_{n(old)}$	0	1	0	0	1	0	1	0	1	0	0	0	0	1	0	0
W_{n+1}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a) The old active wordline $W_{n(old)}$.

W_{n-1}	0	**0**	0	0	**0**	0	0	0	**0**	0	0	0	0	**0**	0	0
$W_{n(new)}$	**0**	**0**	**0**	**0**	**0**	**0**	1	**0**	**0**	**0**	1	0	**0**	**0**	**0**	1
W_{n+1}	0	**0**	0	0	**0**	0	0	0	**0**	0	0	0	0	**0**	0	0

(b) The new active wordline $W_{n(new)}$.

W_{n-1}	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0
$\overline{W_{n(new)}}$	1	1	1	1	1	1	**0**	1	1	1	0	1	1	1	1	0
W_{n+1}	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0

(c) The complement of the new active wordline $\overline{W_{n(new)}}$.

Figure 46: Write disturbance crosstalk in super dense PCM cells. The red and yellow cells represent the aggressor and victim cells, respectively.

the example from Figure 46, not all potential disturbance (crosstalk) can be eliminated. However, if sufficient auxiliary bits are available, rather than setting the bit, the auxiliary bits can point to the location to indicate that it is now storing the incorrect value. Because write disturbance error is asymmetric in PCM, the pointer will always indicate a situation where the status of aggressor cell is kept in the SET state in order to eliminate likelihood of incidence of write disturbance errors. To log each aggressor cell location that was left unchanged, each pointer requires $log_2(512) = 9$ auxiliary bits. *Note that the aggressor cell induces potential write disturbance in the victim cells as long as its status changes from SET state to RESET state ($1 \rightarrow 0$) and idle cells remain in the reset state.*

We compare the number of extra writes required for removing all write disturbance errors (bitline+wordline) for three different approaches, ADAM, four pointers (4pointers), and coset encoding, in Figure 47. ADAM [Swami and Mohanram] requires only one auxiliary bit to indicate whether the cacheline is compressible or not. 4pointers uses $4 \times 9 = 36$ auxiliary bits (7% overhead) to log the cell locations that disturb their neighboring cells. In contrast, coset encoding uses 6.25% additional storage overhead to reduce the number of aggressor cells via increasing the number of codeword candidates. Coset encoding reduces the number of extra writes by 24% and 43% versus 4pointers and ADAM, respectively. ADAM is hampered by only being able to be applied for about 30% cachelines written into memory. As a result, it has the worst performance in our tests. 4pointers is successful in reducing write disturbance by eliminating the cases of highest probability for disturbing potential victim cells in both the active and neighboring wordlines. However, the coset approach has more flexibility to eliminate more potential crosstalk.

The impact of these encoding approaches on endurance (# reset cells) is shown in Figure 48. These tests show that coset encoding also reduces the number of reset cells about 35% and 45% versus 4pointers and ADAM, respectively. Fewer resets will result in longer cell lifetimes. Also, the number of resets for cells directly has an effect on likelihood of disturbance errors. Potential victim cells that have been reset more frequently can more easily crystallize the amorphous state making them more likely to be disturbed over time.

Additionally, using encoding to target the incidence of write disturbance errors can impact the operational energy consumption as shown in Figure 49. Our results show that coset

encoding achieves the minimum energy consumption of the approaches tested. Specifically, it reduces the energy consumption about 18% and 35% versus 4pointers and ADAM, respectively. The multiple codeword candidates of the coset approach for optimizing write disturbance fortunately has positive impacts on both endurance and energy consumption.

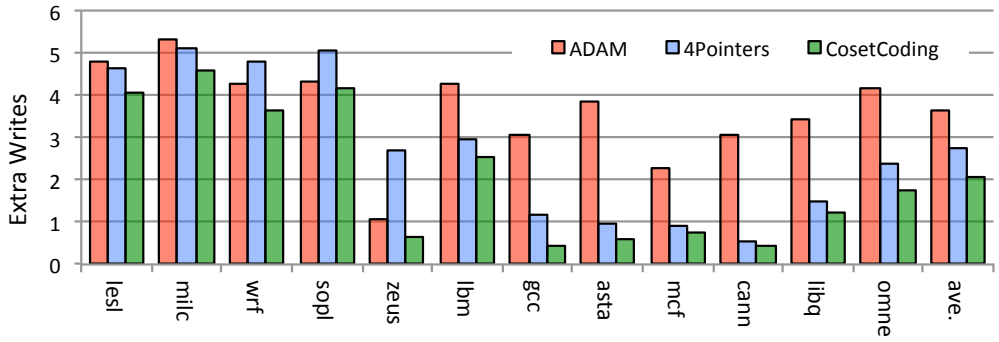


Figure 47: Comparison of extra writes of ADAM, 4pointers and CosetCoding.

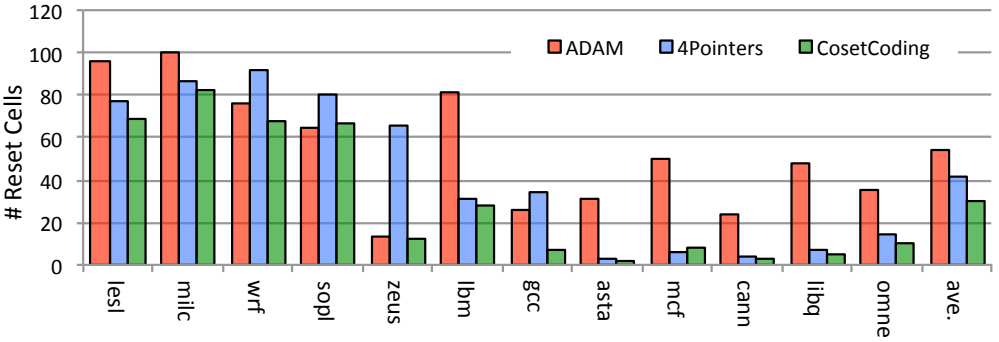


Figure 48: Comparison of endurance of ADAM, 4pointers and CosetCoding.

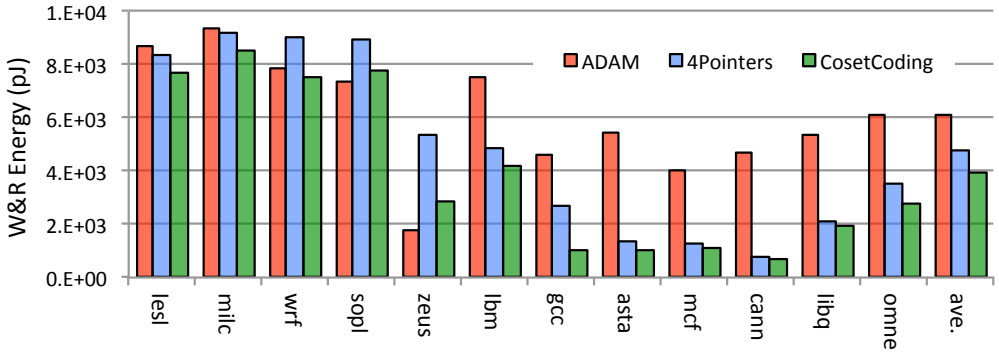


Figure 49: Comparison of energy efficiency of ADAM, 4pointers and Coset Coding.

Both coset and 4pointers incur a significant storage overhead compared to both the baseline and the ADAM technique. These storage overheads can increase the embodied energy by 6-7%, which can dramatically reduce the operational energy benefits from using PCM.

5.3.2 Combined Compression and Encoding

While coset encoding in Section 5.3.1 outperforms ADAM and 4pointers, it incurs 6.25% storage overhead. Our goal is to achieve the same performance (write disturbance mitigation) as coset encoding while incurring no (or at least negligible) area overhead. Note that disturbance errors occur in the data bits and auxiliary bits, therefore removing 6.25% extra area leads to further aggressor cell reduction.

We showed in Section 5.1 that the proposed multi-tiered compression reclaims a small amount of room in 512-bit cachelines, ranging from 40- to 56-bits, without disturbing in-place similarity. Coset encoding with 6.25% area overhead requires only 32 extra bits that can easily be provided by these reclaimed bits. To further reduce disturbance error, we can use remaining reclaimed bits 8- to 24-bits, to eliminate destructive effects of the remaining aggressor cells after encoding through pointers.

The block diagram of our proposed holistic approach is shown in Figure 50. Our proposed approach consists of three main units: the Multi-Tiered Compression (MTC) unit, the Encoder (Enc) unit, and the Disturbance Pointer (DP) unit. During a write, MTC is applied to the 512-bit cacheline. If the cacheline is not compressible, it is directly written into the memory. If the cacheline is compressible, we prioritize the integration of MTC, coset encoding and disturbance pointers as follows.

- $MTC_{[H56]}$: If **MTC** compresses **H**orizontal bits and reclaims **56** bits as shown in Figure 31(a), each word is divided into four sub-words '$D_0 = b_0...b_{15}$', '$D_1 = b_{16}...b_{31}$', '$D_2 = b_{32}...b_{47}$', '$D_3 = b_{48}...b_{55}$'. Each sub-word is encoded by its original data or the complement of original data depending on which one minimizes the number of aggressor cells. Because of the 32 sub-words in the cacheline, it is encoded by 32 reclaimed bits. We use remaining 24 reclaimed bits and supplement with 3 extra auxiliary bits to implement three 9-bit disturbance pointers.

- $MTC_{[V56]}$: If **MTC** compresses **V**ertical bits and reclaims **56** bits as shown in Figure 31(b), each word is similarly divided into four sub-words '$D_0 = b_0...b_{15}$', '$D_1 = b_{16}...b_{31}$', '$D_2 = b_{32}...b_{47}$', '$D_3 = b_{48}...b_{55}$'. The encoding process and addition of three pointers to the design are similar to the $MTC_{[H56]}$. Note that the main difference between $MTC_{[H56]}$ and $MTC_{[V56]}$ is the locations of reclaimed bits and both take advantage of the same type of encoding and the same number of disturbance pointers.

- $MTC_{[H40]}$: If **MTC** compresses **H**orizontal bits and reclaims **40** bits as shown in Figure 31(c), each word is divided into four sub-words '$D_0 = b_0...b_{15}$', '$D_1 = b_{16}...b_{31}$', '$D_2 = b_{32}...b_{47}$', '$D_3 = b_{48}...b_{57}$'. The encoding process is similar to $MTC_{[H56]}$ except that we use the remaining 8 reclaimed bits + 1 extra auxiliary bit to implement a single 9-bit pointer.

- The cacheline is not compressible and the uncompressed cacheline is directly written into the memory.

Using this approach, each memory location requires a total of five additional auxiliary

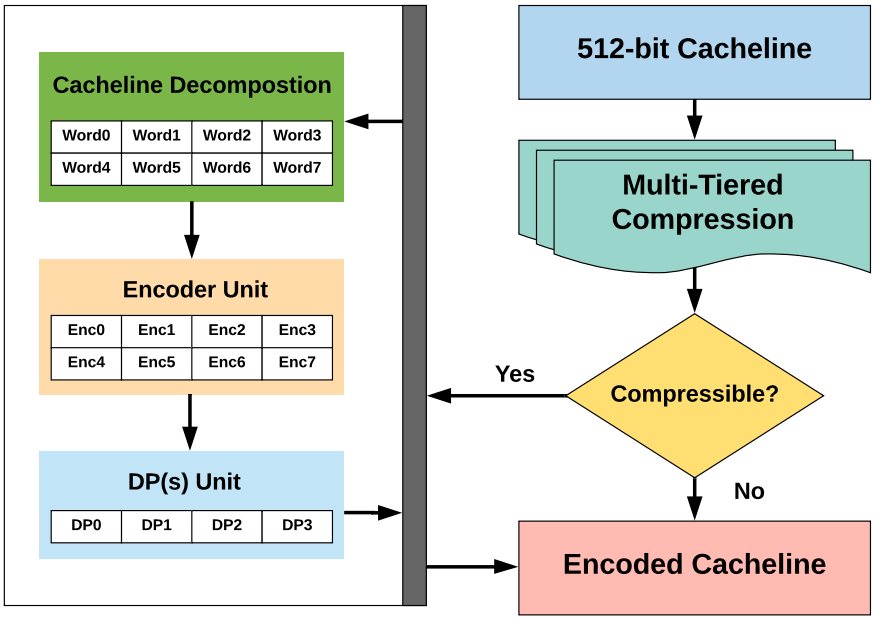


Figure 50: The block diagram of the proposed holistic approach.

bits, where two auxiliary bits are used to indicate which version of MTC is selected by compressor and three auxiliary bits are utilized by $MTC_{[H56]}$ and $MTC_{[V56]}$ for the corresponding disturbance pointers. This results in <1% storage overhead. All eight 64-bit words are encoded in parallel and the corresponding cost function used for each encoder calculates the total probability of incidence of write disturbance errors in the adjacent cells of the aggressor cells. Then, it selects the codeword candidate with the minimum error probability. Since the encoding can not deterministically eliminate all aggressor cells, our combined approach uses the disturbance pointer(s) to log the location of either three (either $MTC_{[H56]}$ or $MTC_{[V56]}$ is used) or one (if $MTC_{[H40]}$ is used) aggressor cell(s) with the highest probability for disturbance due to the SET/RESET state of the neighboring cells.

5.3.3 Evaluation

In this section, we assess the efficiency of various approaches that occupy iso-area. We compare our proposed approach that only uses 5 auxiliary bits per 512-bit cacheline compared to ADAM [Swami and Mohanram] that uses 1 auxiliary bit per 512-bit cacheline (both <1% overhead). We conduct our experiments using the simulator whose configuration is illustrated in Section 5.2.5. According to [Lee et al., 2009], we select the set energy, reset energy and read energy 19.2^{pJ}, 13.5^{pJ}, 2^{pJ}, respectively. Because of bitline and wordline write disturbance errors in super dense PCM, the bit error rates of bitline and wordline write disturbances [Wang et al., 2015] are 9.9% and 11.5%, respectively.

5.3.3.1 Comparison to the State-of-the-art Approach Figure 51 shows the iso-area comparison of our proposed approach and ADAM. For a variety of workloads from high memory intensity workloads such as 'lesl' and 'milc' to low memory intensity workloads such as 'mcf' and 'omne', our proposed approach consistently outperforms ADAM in terms of the extra writes required due to correction of write disturbance errors. Our proposed approach improves efficiency by about 46% versus ADAM via reducing the number of extra writes from 3.67 to 1.98 on average. There are two reasons for this performance improvement. First, while the compression used for our proposed approach is effective for 94% cachelines,

FPC+BDI in ADAM only compresses 30% cachelines. Therefore, ADAM is effective for only a small fraction of cachelines. Second, the integration of coset encoding and the pointer approach reduces the number of aggressor cells that results in the extra write reduction. In contrast, ADAM does not use any encoding and pointers for compressed cachelines and only relies on right and left alignments of compressed cachelines in order to reduce the number of aggressor cells.

Figure 52 illustrates the number of resets as an endurance metric for our approach versus ADAM. Our results show that our proposed approach experiences on average 33 reset cells per write, versus 54 reset cells for ADAM. This results in a 38% endurance improvement. As expected, reducing the number of reset operations also has a positive effect on system performance as it decreases the number of aggressor cells. Figure 53 shows the operational energy efficiency of the proposed approach versus ADAM. Specifically, the proposed approach consumes about 34% less operational energy than ADAM. Recalling that managing write disturbance requires additional reads and potentially rewrites to ensure the data is stored correctly, that the energy reported in this figure encompasses the energy consumption of set and reset operations of the initial write plus these subsequent reads and further rewrites.

5.3.3.2 Multi-Objective Optimization in SLC PCM

The used cost function for the coset coding in Section 5.3.2 minimizes likelihood of incidence of write disturbance errors

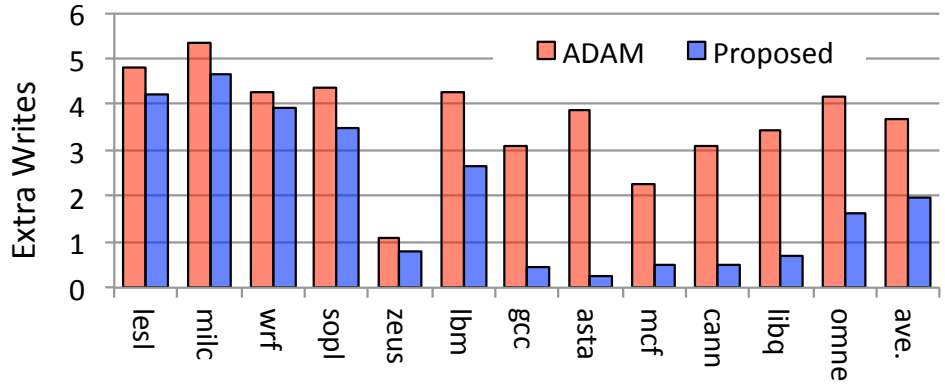


Figure 51: Comparison of extra writes of ADAM and the proposed approach. Both approaches occupy iso-area.

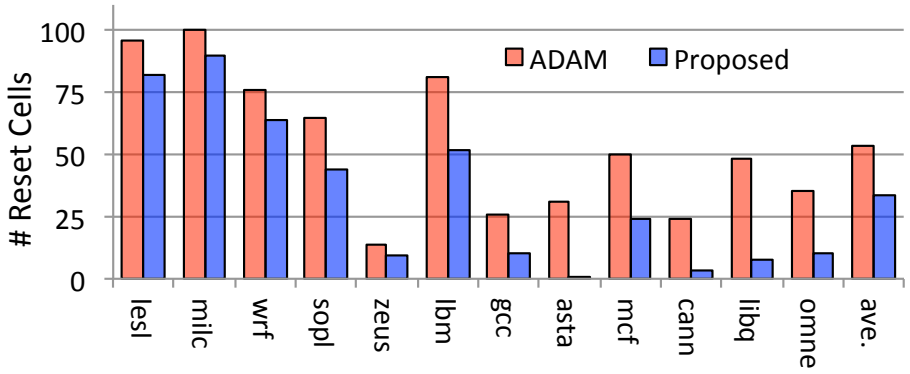


Figure 52: Comparison of # reset cells (endurance) of ADAM and the proposed approach. Both approaches occupy iso-area.

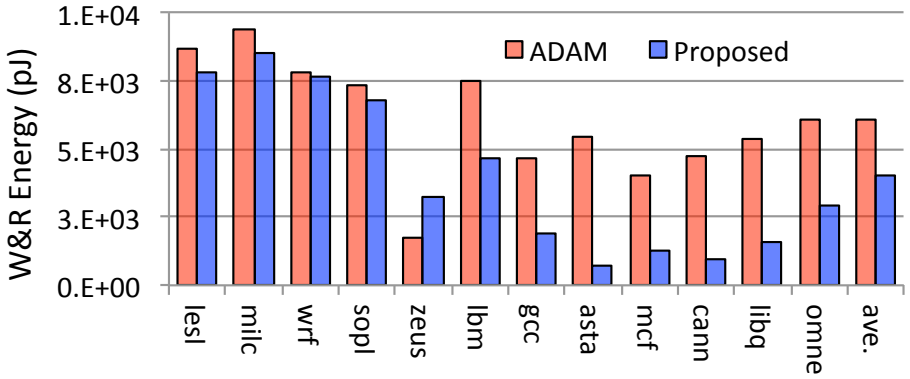


Figure 53: Comparison of energy (write+read) efficiency of ADAM and the proposed approach. Both approaches occupy iso-area.

that leads to extra write reduction. To obtain multi-objective optimization in super dense PCM, we use another cost function that minimizes the number of reset cells. Note that reducing the number of reset cells improves cell endurance in SLC PCM since the reset process reduces PCM cell lifetime. Specifically, we define a disturbance error Threshold (T) that determines whether the cost function optimizes performance or endurance.

Algorithm 3 is a pseudo-code for the multi-objective optimization where the eight 64-bit words, D^i, $i = 0, ..., 7$, are independently encoded in parallel when the memory line is compressible. To encode a word, D^i, it is divided into 4 sub-words D^i_j, $j = 0, ..., 3$ (Lines 1-2), and the sub-words are encoded in parallel using either the original sub-word D^i_j or its complement $\overline{D^i_j}$, depending on the sub-word cost. Note that the write disturbance error

cost D_j^i ($cost_{WDE}(D_j^i)$) is computed by the summation of likelihood of incidence of write disturbance errors (WDE) in D_j^i (Line 3).

As long as the absolute value difference of $cost_{WDE}(D_j^i)$ and $cost_{WDE}(\overline{D_j^i})$ is greater than the disturbance error threshold (T) (Line 5), the codeword with the minimum cost is written into the memory line (Lines 6-7). When the absolute value difference of costs is equal to or less than T, the second cost function is used to minimize the number of reset cells (Line 4). In this case, the cost function computes the number of reset cells for D_j^i and $\overline{D_j^i}$ and then selects the codeword that minimizes the number of reset cells (Lines 8-9).

To analyse sensitivity to the disturbance error threshold, first we select the proposed approach as baseline that only optimizes extra writes and then change T from 0.15 to 1. Our key observation from Figures 54, 55 and 56 is that using the multi-objective optimization on average negligibly changes extra writes and write+read energy while it improves on average endurance 12.5% when the threshold reaches 0.5. For some applications such as 'lesl' that the percentage of bit flips in the memory line is high, the second cost function minimizes the number of transitions ('1 $\rightarrow$ 0' and '0 $\rightarrow$ 1'). In this case, it improves endurance and reduces the total read + write energy. In contrast, for some applications such as 'zeus' that the percentage of bit transitions is not uniform, reducing the number of reset operations

Algorithm 3: Multi-objective optimization applied to a compressible memory line.

```
1 begin
2     for D^i, i = 0, ..., 7, divide D^i into four sub-words D^i_j j = 0, ..., 3 in Parallel do
3         cost_WDE(D^i_j) : ∑ likelihood of incidence of write disturbance errors in D^i_j.
4         cost_#ResetCells(D^i_j) : ∑ reset cells in D^i_j.
5         if |cost_WDE(D^i_j) − cost_WDE(D̄^i_j)| > T then
6             Out = {cost_WDE(D^i_j) < cost_WDE(D̄^i_j)} ? D^i_j : D̄^i_j;
7             /* D̄^i_j is D^i_j complement.*/
8         else
9             Out = {cost_#ResetCells(D^i_j) < cost_#ResetCells(D̄^i_j)} ? D^i_j : D̄^i_j;
```

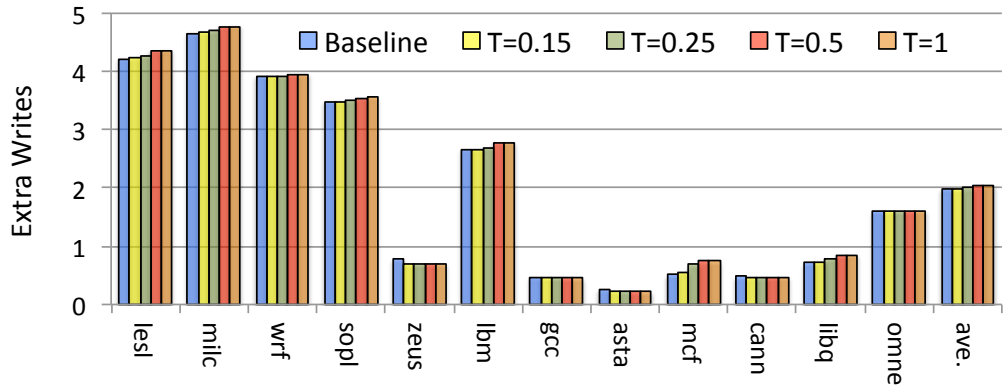


Figure 54: Extra writes of the proposed approach when the threshold changes from 0.15 to 1. Note that the baseline only optimizes the extra writes.

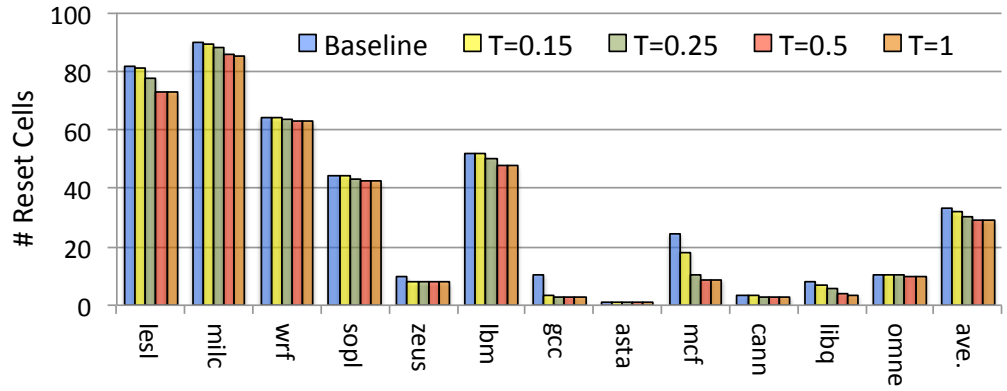


Figure 55: # reset cells of the proposed approach when the threshold changes from 0.15 to 1. Note that the baseline only optimizes the extra writes.

increases the number of set operations that slightly sacrifices write and read energy. However, when T increases from 0.5 to 1, the endurance optimization is saturated and the number of reset and set operations remains unchanged.

5.4 CONCLUSION

Scaling PCM cells below 22nm technology node increases write disturbances among cells in the active wordlines and the corresponding neighboring wordlines that jeopardize reliability of future memory systems. In this chapter, we propose a generic approach that tackles

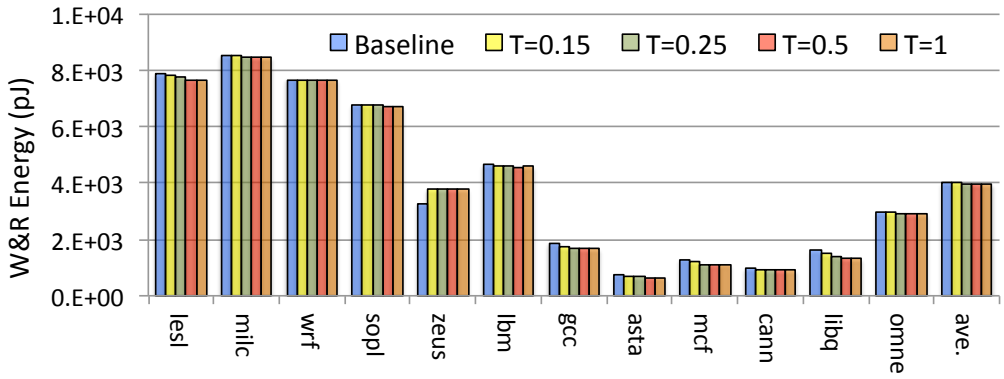


Figure 56: Write+Read energy of the proposed approach when write disturbance threshold changes from 0.15 to 1. Note that the baseline only optimizes the extra writes.

write disturbance crosstalk in super dense PCM cells. We explore byte similarities within and across words in the memory blocks (cachelines) and present a multi-tiered compression (MTC) that can be applied to more than 94% of cachelines in benchmark workloads. We integrate our compression technique to work with an encoding technique that uses cosets and pointers to improve system performance, energy efficiency, and endurance of memory cells without cutting memory capacity.

Our goal in this chapter is to tackle write disturbances in MLC and SLC PCM. In MLC PCM, we reduce the inter-cell spacing along the wordline that eliminates bitline write disturbance errors increasing write energy and wordline write disturbance error rates. We use a special case of MTC, called word level compression (WLC), and integrate with a new Restricted Coset coding to propose WLCRC. The experimental results on real workloads show that WLCRC at 16-bit block granularity improves the write energy by about 52% and 39%, on average, compared to the baseline and the leading write-minimization approach, respectively. It also improves cell endurance and reliability.

To achieve high memory capacity in PCM, the inter-cell spacing along the bitline and wordline is reduced. To tackle both sources (bitline+wordline) of disturbances, we integrate MTC with a coset coding and a disturbance error pointer approach. While coset coding reduces the number of aggressor cells, the pointer approach logs each aggressor cell location that was left unchanged. The experimental tests on realistic workloads show that the proposed approach improves system performance, cell endurance and energy efficiency

about 46%, 38% and 33% versus the state of the art approach while incurring very low area overhead. Furthermore, we improve cell endurance of the proposed approach on average 12.5% via a multi-objective optimization technique without sacrificing energy efficiency, performance and memory capacity.

6.0 SUMMARY AND CONCLUSION OF THE THESIS

In data deluge era, 2.5 quintillion bytes trillion Gigabytes of data are created every day [Dobre and Xhafa, 2014]. This high volume of data forces increasingly insistent demands for many core systems. As the number of cores per chip continues to increase, the need for a large memory capacity that serves the requests of the executing cores is pressing more than ever. Scaling down process technology enables higher memory capacity through reducing the size and proximity of memory cells. Unfortunately, this trend jeopardizes current memory designs, especially when scaling beyond 22nm technology node, because of fundamental obstacles related to voltage fluctuations and process variation problems. For example, when the cumulative interference to a DRAM wordline becomes strong enough due to technology scaling, the state of nearby cells can change leading to *inter-cell read disturbance errors*.

Technology scaling in promising technologies like STT-RAM limits thermal stability and increases accumulated read current pulses. In this case, a *intra-cell read disturbance* accidentally flips the value stored within a cell resulting in subsequent read errors that persist until a new value is written into the cell. For technology nodes below 22nm in super dense PCM, the heat used during the writing process bleeds to neighboring cells and leads to inadvertent writing, referred to as *write disturbance errors*. Therefore, tackling disturbance errors to guarantee memory reliability is a key concern to enable technology scaling for building high-density memory chips. To address these concerns, this dissertation suggests three broad designs and techniques over three chapters.

Chapter 3 introduces a tree-based non-uniform row partitioning for tackling read disturbance errors in DRAM banks. It develops a low-cost implementation with three key ideas: (1) A low-cost implementation to maintain and access Counter-based Adaptive Trees that assign counters to rows non-uniformly and detects more precisely victim rows. (2) A scheme

to compute the split thresholds that cause the trees to dynamically evolve and match the row access patterns. (3) A scheme, DRCAT, for dynamically reconfiguring the CAT to track the temporal changes in memory access patterns resulting from either changing the running applications or changing the phases of a running application. The experimental results show that DRCAT outperforms the leading approaches for wordline disturbance mitigation. Specifically, for quad-core systems and refresh threshold of T = 16K, DRCAT reduces the power overhead to 7%, which is an improvement over the 21% and 18% incurred in deterministic and probabilistic approaches, respectively. Moreover, DRCAT incurs very low performance overhead (<0.5%).

Chapter 4 studies three approaches to mitigate read disturbances for STT-RAM compared to the conservative approach of writing back after every read (WAR). These approaches leverage a single ECC to cover all three different types of faults/errors. In particular, the schemes are proposed to write back blocks after any error is detected (WAE), after a persistent error (due to read disturbance or write fault) is detected (WAP), or after multiple errors are detected (WAT). Further, a Markov modeling approach is provided to evaluate all three types of errors and generate a single reliability of the system in terms of uncorrectable bit error rate. Moreover, an energy reliability product metric is described to be able to quantitative evaluate the trade-off between system energy and reliability. In summary, it is shown qualitatively that WAE, WAP and WAT provide dramatic improvement in energy consumption and memory bandwidth overhead (due to additional write-backs) while eliminating the effects of read disturbance and retaining near WAR reliability.

Chapter 5 explores byte similarities within and across words in the memory blocks (cachelines) and presents a Multi-Tiered Compression (MTC) that can be applied to more than 94% of cachelines in benchmark workloads. Also, it proposes a generic approach that integrates MTC with the coset coding in order to optimize cost functions based on write energy reduction and performance improvement while not sacrificing memory capacity. To this end, this chapter tries to deal with bitline and wordline write disturbances in SLC and MLC PCM.

Given the reduction of inter-cell spacing along the wordline, we combat wordline write disturbance errors. To increase memory capacity, MLC PCM that suffers from high write

energy and high wordline bit error rates is taken into account. A novel restricted coset encoding is proposed that largely reduces the number of auxiliary bits compared to known coset encodings while achieving similar write energy reduction. Furthermore, a Word Level Compression (WLC) technique is used that compresses 90% of the memory blocks while reclaiming enough space in the compressed lines to fit the auxiliary bits. Finally, a new and low hardware overhead architecture, WLCRC, is presented that integrates WLC and restricted coset encoding to effectively reduce the write energy in MLC PCM. Hardware synthesis indicates that WLCRC encoders and decoders incur low area, latency, and energy overheads. Our experimental results on real workloads show that WLCRC at 16-bit block granularity improves the write energy by about 52% and 39%, on average, compared to the baseline and the leading write-minimization approach, respectively. It also improves cell endurance and reliability although no specific provisions are made during the encoding to optimize these metrics.

Given the reduction of inter-cell spacing along the wordline and the bitline, we also combat both bitline and wordline write disturbances in super dense PCM. We utilize MTC technique to work with an encoding technique that uses cosets and pointers to improve system performance, cell endurance, and energy efficiency of single level memory cells. The experimental tests on realistic workloads show that the proposed approach improves system performance, cell endurance and energy efficiency about 46%, 38% and 33% versus the state of the art approach while incurring very low area overhead. Furthermore, by multi-objective optimization, cell endurance of the proposed approach is improved on average 12.5% while not sacrificing energy, performance and memory capacity.

While the proposed generic approach in Chapter 5 tackles high programming energy and high wordline disturbance error rates in MLC PCM and also bitline+wordline disturbances in SLC PCM, it can be reconciled with various objectives for other memory technologies. For example, while flash memories [Berman and Birk, 2012; Buzaglo and Siegel, 2017] are one of the most important types of non-volatile memories, still they suffer inter-cell interferences that are data dependent. Specifically, when data patterns '101' appear in the bitlines and wordlines of the flash memory, the voltage level of the victim cell thanks to parasitic capacitances increases and the cell state changes from 0 to 1. The data-dependency and

uni-directionality of errors provide opportunity for the cost function in the restricted coset coding to diminish likelihood of incidence of data patterns '101' while improving flash memory reliability. Our generic approach also can be used for reducing asymmetric transmission costs in the I/O memory bus [Song and Ipek; Wang and Ipek] whose energy consumption is correlated to the type of symbols transmitted over long and highly capacitive interconnects.

Finally, we conclude that as memory technology scales in size, the goals of reliability, energy efficiency, performance and security often clash with one another. Leveraging practical hardware techniques to efficiently and accurately resolve vulnerabilities in future memory systems is a promising strategy to adjust these conflicting objectives to the correct pitch. The work in this dissertation identifies the root of forthcoming critical challenges in future memory systems and eradicates it through low-overhead architectural techniques.

BIBLIOGRAPHY

https://users.ece.cmu.edu/ koopman/lfsr/.

4Gb DDR3 SDRAM, 2011. 2011.

Nidhi Aggarwal, Jason F Cantin, Mikko H Lipasti, and James E Smith. Power-efficient dram speculation. In *HPCA 2008*.

Su Jin Ahn, Yoonjong Song, Hoon Jeong, Byeungchul Kim, Youn-Seon Kang, Dong-Ho Ahn, Yongwoo Kwon, Seok Woo Nam, Gitae Jeong, Hokyu Kang, et al. Reliability perspectives for high density pram manufacturing. In *IEDM 2011*.

Alaa R Alameldeen and David A Wood. Frequent pattern compression: A significance-based compression scheme for l2 caches. *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, 1500, 2004.

Mohammad Arjomand, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Boosting access parallelism to pcm-based main memory. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 695–706, 2016.

Manu Awasthi and et al. Efficient scrub mechanisms for error-prone emerging memories. In *HPCA 2012*.

Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. Anvil: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices*, 51(4):743–755, 2016.

Kuljit S Bains and John B Halbert. Distributed row hammer tracking, March 29 2016. US Patent 9,299,400.

Kuljit S Bains, John B Halbert, Christopher P Mozak, Theodore Z Schoenborn, and Zvika Greenfield. Row hammer refresh command, January 12 2016. US Patent 9,236,110.

Saisanthosh Balakrishnan and Gurindar S Sohi. Exploiting value locality in physical register files. In *MICRO 2003*.

Ferdinando Bedeschi, Rich Fackenthal, Claudio Resta, Enzo Michele Donze, Meenatchi Jagasivamani, Egidio Cassiodoro Buda, Fabio Pellizzer, David W Chow, Alessandro Cabrini,

Giacomo Matteo Angelo Calvi, et al. A bipolar-selected phase change memory featuring multi-level cell storage. *IEEE Journal of Solid-State Circuits*, 44(1):217–227, 2009.

Amit Berman and Yitzhak Birk. Constrained flash memory programming. In *ISIT 2011*.

Amit Berman and Yitzhak Birk. Low-complexity two-dimensional data encoding for memory inter-cell interference reduction. In *IEEEI*, pages 1–5, 2012.

Ishwar Bhati, Mu-Tien Chang, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. Dram refresh mechanisms, penalties, and trade-offs. *IEEE Transactions on Computers*, 65(1):108–121, 2016.

Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *SP 2016*.

Stefania Braga, Alessandro Sanasi, Alessandro Cabrini, and Guido Torelli. Voltage-driven partial-reset multilevel programming in phase-change memories. *IEEE Transactions on Electron Devices*, 57(10):2556–2563, 2010.

Sarit Buzaglo and Paul H Siegel. Row-by-row coding schemes for inter-cell interference in flash memory. *IEEE Transactions on Communications*, 65(10):4101–4113, 2017.

Yu Cai and et al. Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery. In *DSN 2015*.

Yu Cai, Gulay Yalcin, and et al. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *ICCD 2012*.

SY Cha. Dram and future commodity memories. *VLSI Technology Short Course*, 2011.

Niladrish Chatterjee, Rajeev Balasubramonian, Manjunath Shevgoor, Seth Pugsley, Aniruddha Udipi, Ali Shafiee, Kshitij Sudan, Manu Awasthi, and Zeshan Chishti. Usimm: the utah simulated memory module. *University of Utah, Tech. Rep*, 2012.

E Chen and et al. Advances and future prospects of spin-transfer torque random access memory. *IEEE Transactions on Magnetics*, 2010.

Sangyeun Cho and Hyunjin Lee. Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *MICRO 2009*.

Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *DAC 2011*.

Ciprian Dobre and Fatos Xhafa. Intelligent services for big data science. *Future Generation Computer Systems*, 37:267–281, 2014.

Xiangyu Dong and Yuan Xie. Adams: Adaptive mlc/slc phase-change memory design for file storage. In *ASP-DAC 2011*.

Magnus Ekman and Per Stenstrom. A robust main-memory compression scheme. In *ACM SIGARCH Computer Architecture News*, volume 33, pages 74–85, 2005.

G David Forney. Coset codes. i. introduction and geometrical classification. *IEEE Transactions on Information Theory 1988*.

FreePDK45. http://www.eda.ncsu.edu/wiki/.

Mohsen Ghasempour, Mikel Lujan, and Jim Garside. Armor: A run-time memory hot-row detector, 2015.

Mohsen Ghasempour, Mikel Lujan, and Jim Garside. Armor: A run-time memory hot-row detector., http://apt.cs.manchester.ac.uk/projects/ARMOR /RowHammer/ index.html.

Mrinmoy Ghosh and Hsien-Hsin S Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams. In *MICRO 2007*.

Zvika Greenfield, Kuljit S Bains, Theodore Z Schoenborn, Christopher P Mozak, and John B Halbert. Row hammer condition monitoring, January 20 2015. US Patent 8,938,573.

Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 2012.

Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer. js: A remote software-induced fault attack in javascript. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 300–321. Springer, 2016.

N Herath and A Fogh. These are not your grand daddy's cpu performance counters: Cpu hardware performance counters for security. *Black Hat 2015*.

Mark Horowitz, Thomas Indermaur, and Ricardo Gonzalez. Low-power digital design. In *IEEE Symposium on Low Power Electronic 1994*.

M Hosomi, H Yamagishi, T Yamamoto, K Bessho, Y Higo, K Yamane, H Yamada, M Shoji, H Hachino, C Fukumoto, et al. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In *IEDM 2005*.

Bruce Jacob and et al. *Memory systems: cache, DRAM, disk*. 2010.

Adam N Jacobvitz, Robert Calderbank, and Daniel J Sorin. Coset coding to extend the lifetime of memory. In *HPCA 2013*.

Min Kyu Jeong, Doe Hyun Yoon, Dam Sunwoo, Mike Sullivan, Ikhwan Lee, and Mattan Erez. Balancing dram locality and parallelism in shared memory cmp systems. In *HPCA 2012*.

Lei Jiang, Youtao Zhang, and Jun Yang. Mitigating write disturbance in super-dense phase change memories. In *DSN 2014*.

Madhura Joshi, Wangyuan Zhang, and Tao Li. Mercury: A fast and energy-efficient multi-level cell based phase change memory system. In *HPCA 2011*.

Wang Kang, WeiSheng Zhao, Zhaohao Wang, Yue Zhang, Jacques-Olivier Klein, Youguang Zhang, Claude Chappert, and Dafiné Ravelosona. A low-cost built-in error correction circuit design for stt-mram reliability improvement. *Microelectronics Reliability*, 53(9): 1224–1229, 2013.

Takayuki Kawahara and et al. 2mb spin-transfer torque ram (spram) with bit-by-bit bidirectional current write and parallelizing-direction current read. In *ISSCC 2007*.

Mazen Kharbutli and Yan Solihin. Counter-based cache replacement algorithms. In *ICCD 2005*.

Byeungchul Kim, Yoonjong Song, Sujin Ahn, Younseon Kang, Hoon Jeong, Dongho Ahn, Seokwoo Nam, Gitae Jeong, and Chilhee Chung. Current status and future prospect of phase change memory. In *ASIC*, pages 279–282, 2011.

Dae-Hyun Kim, Prashant J Nair, and Moinuddin K Qureshi. Architectural support for mitigating row hammering in dram memories. *IEEE Computer Architecture Letters*, 14 (1):9–12, 2015.

Joohee Kim and Marios C Papaefthymiou. Block-based multiperiod dynamic memory design for low data-retention power. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(6):1006–1018, 2003.

Jungrae Kim, Michael Sullivan, Esha Choukse, and Mattan Erez. Bit-plane compression: Transforming data for better compression in many-core architectures. In *ISCA 2016*, a.

Jungrae Kim, Michael Sullivan, Seong-Lyong Gong, and Mattan Erez. Frugal ecc: Efficient and versatile memory error protection through fine-grained compression. In *International Conference for High Performance Computing, Networking, Storage and Analysis 2015*, b.

Kinam Kim. Technology for sub-50nm dram and nand flash manufacturing. In *IDEM 2005*.

Kinarn Kim and Su Jin Ahn. Reliability investigations for manufacturable high density pram. In *Reliability Physics 2005*.

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372, 2014.

John F Kitchin. Practical markov modeling for reliability analysis. In *Reliability and Maintainability Symposium, 1988*.

RH Koch and et al. Time-resolved reversal of spin-transfer switching in a nanomagnet. *Physical review letters*, 2004.

Jagadish B Kotra, Narges Shahidi, Zeshan A Chishti, and Mahmut T Kandemir. Hardware-software co-design to mitigate dram refresh overheads: a case for refresh-aware process scheduling. In *ASPLOS 2017*.

Emre Kultursay and et al. Evaluating stt-ram as an energy-efficient main memory alternative. In *ISPASS 2013*.

Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 2–13, 2009.

SH Lee, MS Kim, GS Do, SG Kim, HJ Lee, JS Sim, NG Park, SB Hong, YH Jeon, KS Choi, et al. Programming disturbance and cell scaling in phase change memory: For up to 16nm based 4f 2 cell. In *VLSIT 2010*.

Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. Flikker: saving dram refresh-power through critical data partitioning. *ACM SIGPLAN Notices*, 47 (4):213–224, 2012.

Peter S Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

Jack A Mandelman, Robert H Dennard, Gary B Bronner, John K DeBrosse, Rama Divakaruni, Yujun Li, and Carl J Radens. Challenges and future directions for the scaling of dynamic random-access memory (dram). *IBM Journal of Research and Development 2002*.

howpublished = http://www.cs.utah.edu/rajeev/jwac12/ Memory Scheduling Championship.

Justin Meza, Jing Li, and Onur Mutlu. Evaluating row buffer locality in future non-volatile main memories. 2012.

Asit K Mishra, Xiangyu Dong, Guangyu Sun, Yuan Xie, Narayanan Vijaykrishnan, and Chita R Das. Architecting on-chip interconnects for stacked 3d stt-ram caches in cmps. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 69–80, 2011.

Janani Mukundan, Hillery Hunter, Kyu-hyoun Kim, Jeffrey Stuecheli, and José F Martínez. Understanding and mitigating refresh overheads in high-density ddr4 dram systems. *ACM SIGARCH Computer Architecture News*, 41(3):48–59, 2013.

Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to understand large caches. *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, 2009.

Helia Naeimi, Charles Augustine, Arijit Raychowdhury, Shih-Lien Lu, and James Tschanz. Sttram scaling and retention failure. *Intel Technology Journal*, 17(1), 2013.

Prashant Nair, Chia-Chen Chou, and Moinuddin K Qureshi. A case for refresh pausing in dram memory systems. In *HPCA 2013*.

T Nirschl, JB Philipp, TD Happ, Geoffrey W Burr, B Rajendran, M-H Lee, A Schrott, M Yang, M Breitwisch, C-F Chen, et al. Write strategies for 2 and 4-bit multi-level phase-change memory. In *IEDM 2007*.

Taku Ohsawa, Koji Kai, and Kazuaki Murakami. Optimizing the dram refresh count for merged dram/logic lsis. In *ISLPED 1998*.

A Pantazi, A Sebastian, N Papandreou, MJ Breitwisch, C Lam, H Pozidis, and E Eleftheriou. Multilevel phase change memory modeling and experimental characterization. *Proceedings of EPCOS*, 2009.

Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Linearly compressed pages: a low-complexity, low-latency main memory compression framework. In *MICRO 2013*.

Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Base-delta-immediate compression: practical data compression for on-chip caches. In *PACT*, pages 377–388. ACM, 2012.

Amir Rahmati, Matthew Hicks, Daniel Holcomb, and Kevin Fu. Refreshing thoughts on dram: Power saving vs. data integrity. In *WACAS 2014*.

Arijit Raychowdhury. Pulsed read in spin transfer torque (stt) memory bitcell for lower read disturb. In *NANOARCH 2013*.

Michael Redeker, Bruce F Cockburn, and Duncan G Elliott. An investigation into crosstalk noise in dram structures. In *MTDT 2002*.

Eric Rotenberg. Ar-smt: A microarchitectural approach to fault tolerance in microprocessors. In *Fault-Tolerant Computing 1999*.

Toshinori Sato and Toshimasa Funaki. Dependability, power, and performance trade-off on a multicore processor. In *ASP-DAC 2008*.

Luca Schiano, Marco Ottavi, and Fabrizio Lombardi. Markov models of fault-tolerant memory systems under seu. In *MTDT 2004*.

Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. *Black Hat 2015*.

Hoseok Seol, Wongyu Shin, Jaemin Jang, Jungwhan Choi, Jinwoong Suh, and Lee-Sup Kim. Energy efficient data encoding in dram channels exploiting data value similarity. In *ISCA 2016*.

Seyed Mohammad Seyedzadeh, Rakan Maddah, Alex Jones, and Rami Melhem. Pres: Pseudo-random encoding scheme to increase the bit flip reduction in the memory. In *DAC 2015*.

Seyed Mohammad Seyedzadeh, Rakan Maddah, Alex Jones, and Rami Melhem. Leveraging ecc to mitigate read disturbance, false reads and write faults in stt-ram. In *DSN*, pages 215–226, 2016a.

Seyed Mohammad Seyedzadeh, Rakan Maddah, Donald Kline, Alex K Jones, and Rami Melhem. Improving bit flip reduction for biased and random data. *IEEE Transactions on Computers*, 65(11):3345–3356, 2016b.

SeyedMohammad Seyedzadeh, Alex Jones, and Rami Melhem. Enabling fine-grain restricted coset coding through word-level compression for pcm. In *HPCA*, pages 350–361, 2018.

Wongyu Shin, Jungwhan Choi, Jaemin Jang, Jinwoong Suh, Youngsuk Moon, Yongkee Kwon, and Lee-Sup Kim. Dram-latency optimization inspired by relationship between row-access time and refresh timing. *IEEE Transactions on Computers*, 65(10):3027–3040, 2016.

Kirill A Shutemov. Pagemap: Do not leak physical addresses to non-privileged userspace. *Retrieved on November*, 10:2015, 2015.

Padhraic Smyth. Hidden markov models for fault detection in dynamic systems. *Pattern recognition*, 27(1):149–164, 1994.

Yanwei Song and Engin Ipek. More is less: Improving the energy efficiency of data movement via opportunistic use of sparse codes. In *MICRO 2015*.

Suresh Srinivasan, Sanu Mathew, Rajaraman Ramanarayanan, Farhana Sheikh, Mark Anders, Himanshu Kaul, Vasantha Erraguntla, Ram Krishnamurthy, and Greg Taylor. 2.4 ghz 7mw all-digital pvt-variation tolerant true random number generator in 45nm cmos. In *VLSIC 2010*.

Zhenyu Sun, Hai Li, and Wenqing Wu. A dual-mode architecture for fast-switching stt-ram. In *ISLPED 2012*.

Shivam Swami and Kartik Mohanram. Adam: Architecture for write disturbance mitigation in scaled phase change memory. In *DATE 2018*.

R Takemura and et al. Highly-scalable disruptive reading scheme for gb-scale spram and beyond. In *IMW 2010*.

William Turin and M. Mohan Sondhi. Modeling error sources in digital channels. *IEEE Journal on Selected Areas in Communications*, 11(3):340–347, 1993.

Ad J Van De Goor and Ivo Schanstra. Address and data scrambling: Causes and impact on memory tests. In *Proceedings First IEEE International Workshop on Electronic Design, Test and Applications 2002*.

Jue Wang, Xiangyu Dong, Guangyu Sun, Dimin Niu, and Yuan Xie. Energy-efficient multi-level cell phase-change memory system with data encoding. In *ICCD 2011*.

Rujia Wang and et al. Selective restore: an energy efficient read disturbance mitigation scheme for future stt-mram. In *DAC 2015*.

Rujia Wang, Lei Jiang, Youtao Zhang, and Jun Yang. Sd-pcm: Constructing reliable super dense phase change memory under write disturbance. *ACM SIGARCH Computer Architecture News*, 43(1):19–31, 2015.

Shibo Wang and Engin Ipek. Reducing data movement energy via online data clustering and encoding. In *MICRO 2016*.

Wujie Wen, Yaojun Zhang, Yiran Chen, Yu Wang, and Yuan Xie. Ps3-ram: A fast portable and scalable statistical stt-ram reliability analysis method. In *DAC 2012*.

Chengen Yang and et al. Improving reliability of non-volatile memory technologies through circuit level techniques and error control coding. *EURASIP*, 2012.

Jun Yang, Youtao Zhang, and Rajiv Gupta. Frequent value compression in data caches. In *MICRO*, pages 258–265, 2000.

Kaiyuan Yang, David Fick, Michael B Henry, Yoonmyung Lee, David Blaauw, and Dennis Sylvester. 16.3 a 23mb/s 23pj/b fully synthesized true-random-number generator in 28nm and 65nm cmos. In *ISSCC 2014*.

Wangyuan Zhang and Tao Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *MICRO 2009*.

Yaojun Zhang, Ismail Bayram, Yu Wang, Hai Li, and Yiran Chen. Adams: Asymmetric differential stt-ram cell structure for reliable and high-performance applications. In *ICCAD 2013*.

Yaojun Zhang, Wujie Wen, and Yiran Chen. The prospect of stt-ram scaling from readability perspective. *IEEE Transactions on Magnetics*, 48(11):3035–3038, 2012.

Yaojun Zhang, Yong Li, Zhenyu Sun, Hai Li, Yiran Chen, and Alex K Jones. Read performance: The newest barrier in scaled stt-ram. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(6):1170–1174, 2015.

Youtao Zhang, Jun Yang, and Rajiv Gupta. Frequent value locality and value-centric data cache design. *ACM SIGPLAN Notices*, 35(11):150–159, 2000.

WS Zhao and et al. Failure and reliability analysis of stt-mram. *Microelectronics Reliability*, 2012.

Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *ACM SIGARCH computer architecture news*, 2009.